Neuromorphic Autopilot for Drone Flight

Stroobants, S.

DOI
10.4233/uuid:4e8503ea-d09b-4554-b5a0-d6d68e2c6dfd

Publication date
2025

Document Version
Final published version

Citation (APA)
Stroobants, S. (2025). *Neuromorphic Autopilot for Drone Flight*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:4e8503ea-d09b-4554-b5a0-d6d68e2c6dfd

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Neuromorphic Autopilot for Drone Flight

Stein Stroobants

Neuromorphic Autopilot for Drone Flight

Neuromorphic Autopilot for Drone Flight

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates
to be defended publicly on
Wednesday 3 December 2025 at 15:00 o'clock

by

Stein STROOBANTS

Master of Science in Systems and Control,
Delft University of Technology, The Netherlands,
born in Amsterdam, The Netherlands

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. G.C.H.E. de Croon,	Delft University of Technology, *promotor*
Dr. ir. C. De Wagter,	Delft University of Technology, *copromotor*
Independent members:	
Prof. dr. L.L.A. Vermeersen,	Delft University of Technology
Prof. dr. ir. T. Keviczky,	Delft University of Technology
Prof. dr. Y. Sandamirskaya,	Zürich University of Applied Sciences
dr. C.D. Schuman,	University of Tennessee Knoxville
dr. G. Spigler,	Tilburg University

Keywords: Neuromorphic Computing, Artificial Neural Networks, Low-level Control, Drones, Event-based Cameras, Spiking Neural Networks, Supervised Learning, State Estimation

Printed by: Ipskamp Printing

Cover by: J.C. Steinhoff

ISBN 978-94-6384-870-1

An electronic copy of this dissertation is available at
`https://repository.tudelft.nl/`.

The men of experiment are like the ant, they only collect and use; the reasoners resemble spiders, who make cobwebs out of their own substance. But the bee takes a middle course: it gathers its material from the flowers of the garden and of the field, but transforms and digests it by a power of its own.

Sir Francis Bacon, *Novum Organum*

Every new discovery is just a reminder—we're all small and stupid

Evelyn Wang, *Everything Everywhere All at Once*

Contents

Summary

There exists a wide array of possible applications for small, safe, cost-effective, and energy-efficient drones. However, their development is hampered by limited payload capacity, which restricts both computational power and flight time. Traditional control systems and sensor processing algorithms are ill-suited for these resource-constrained platforms since they typically rely on power-hungry processors and complex numerical methods.

This thesis investigates neuromorphic approaches to both state estimation and control for small drones. Inspired by the energy-efficient and highly parallel processing of biological neural systems, neuromorphic computing leverages spiking neural networks (SNNs) that operate via discrete spikes, offering real-time, low-power processing capabilities for micro aerial vehicles (MAVs). While previous work has applied neuromorphic methods to high-level perception tasks, their application to fundamental flight control – such as precise attitude estimation and low-level control – remains largely unexplored.

Following a review of the current state of neuromorphic computing, the research first explores its application to state estimation. A recurrent SNN is designed to estimate the drone's attitude from inertial measurement unit (IMU) data, achieving performance comparable to conventional methods like the complementary filter, despite employing a minimal network architecture. The study then investigates event-based vision sensors by processing data from a downward-facing event camera to estimate the attitude and angular rates, enabling a quadrotor to achieve flight without inertial sensing – a pioneering demonstration in the field.

Transitioning from estimation to control, the thesis uses neuromorphic algorithms to perform low-level control tasks. A spiking PID controller is developed using a fixed network architecture, demonstrating altitude control using Intel's Loihi neuromorphic processor. To address the challenge of precise integration inherent in spiking systems, the Input-Weighted Threshold Adaptation (IWTA) mechanism is introduced. This innovative approach allows for precise integration of incoming signals and was used as the integral component of a

neuromorphic PID controller, mitigating steady-state errors and compensating for sensor biases.

Ultimately, the work unifies estimation and control into a single end-to-end neuromorphic system deployed on a tiny 27g Crazyflie quadrotor. Trained via imitation learning on real flight data, the integrated network maps raw inertial sensor inputs directly to motor commands at a control frequency of 500Hz, achieving attitude tracking performance comparable to traditional controllers.

Overall, this thesis demonstrates that neuromorphic computing is a promising approach for low-level state estimation and control in flying drones, while also addressing the challenges of implementing such systems in real-world environments with sensor biases and persistent disturbances.

1

Introduction

The versatility of drones is unparalleled — they require minimal infrastructure, can adapt to diverse and unstructured environments, and perform an array of tasks on a single platform. Drones have demonstrated the ability to address pressing challenges in fields as diverse as agriculture, disaster response, infrastructure inspection, and environmental monitoring. For instance, drones are used to survey crops with precision, inspect power lines and railways, and even collect atmospheric data for climate studies (Mohsan *et al.* [1]; Faiçal *et al.* [2]). At the heart of drone technology lies a vision to enhance human society by extending human capabilities. Drones can reduce risk by performing tasks in hazardous conditions, such as search-and-rescue operations in disaster-stricken areas, monitoring active volcanoes, or delivering medical supplies to remote locations (Lyu *et al.* [3]). Additionally, drones are uniquely positioned to assume jobs that are monotonous, labor-intensive, or undesirable, such as transporting medical supplies or large-scale inspection tasks. For drones to become viable alternatives in these scenarios, however, two key challenges must be addressed: autonomy and safety. Autonomous drones must be capable of executing complex tasks without human intervention, while safety requirements necessitate designs that prevent harm to both humans and the environment.

1.1. The Case for Smaller Drones

Drones are often associated primarily with military applications, which can obscure their vast potential for positive societal impact. In recent years, there has been growing interest in the development of smaller drones, which offer distinct advantages over their larger counterparts. Smaller drones are inherently safer, as their low mass and small rotors reduce the risk of injury in the event of a collision (Floreano *et al.* [4]). These characteristics make them suitable for applications near humans, such as assisting in indoor tasks, surveying sensitive environments, or exploring confined spaces. A potential use-case is found in flapping-wing drones (de Croon [5], see Figure 1.1). Remarkably, these bio-inspired robots can safely be deployed in greenhouses without the risk of harm to farmers and crop[1]. Also, they are more energy efficient at smaller size than drones with rotating propellors (Hawkes *et al.* [6]), and they can perform agile maneuvers (Karásek *et al.* [7]).

[1] Flapper (`www.flapper-drones.com`) in a greenhouse: `www.youtube.com/watch?v=90Y8eRSLp7Q`

Economically, smaller drones are more cost-effective than their larger counterparts. Their lower production costs allow for greater scalability, enabling the deployment of swarms or fleets to perform collaborative tasks, such as crop health monitoring or wildlife tracking (Zhou *et al.* [8]). The potential of swarms of small-sized drones has already been demonstrated in use cases such as search-and-rescue (McGuire *et al.* [9]) and gas detection (Duisterhof *et al.* [10], see Figure 1.1). Furthermore, their affordability reduces the stakes associated with failure, allowing dispensable drones to operate in high-risk scenarios that might otherwise be prohibitive.

Figure 1.1: Small drones can operate in high-risk scenarios. A group of tiny quadrotors (*left*) collaborating to find a gas source [10]. Flapping wing drones (*right*) can safely operate around humans.

Despite these advantages, all small drones face several limitations. Their small form factor restricts payload capacity, limiting the size and power of sensors, processors, and batteries that can be integrated. This, in turn, imposes constraints on their autonomy, as traditional computational approaches and sensing technologies cannot be directly scaled down to match the size and energy constraints of these platforms. To overcome these challenges, a paradigm shift is required in how these drones process information and perform tasks.

1.2. The Neuromorphic Paradigm

The increasing interest in autonomous small drones poses a fundamental challenge: achieving real-time, adaptive control under the constraints of limited computational resources, power efficiency, and environmental unpredictability. Traditional control architectures rely on model-based approaches or machine learning techniques that demand high-performance computation – often impractical for embedded drone platforms. If we look at nature's efficient

designs, for instance in small flying insects such as honeybees, fruit flies, and cicadas, we observe that biological systems outperform our engineered solutions in energy expenditure, sensory processing, and robustness during flight. Despite significant technological advances, we have yet to fully understand or replicate these natural strategies. How can a bee perform complex behaviors such as path-finding and communication in flight with far less power and weight than an embedded system based on an STM32F4 microcontroller such as the tiny Crazyflie, a 25 gram quadrotor designed by Bitcraze[2]? In this thesis, we investigate methods to bridge this computational gap. Specifically, we study neuromorphic computing, which moves away from synchronous and dense processing in favor of biologically inspired asynchronous and sparse method of information handling (Schuman *et al.* [11]; Davies *et al.* [12]). Nature has evolved remarkably efficient solutions for processing information, and drawing inspiration from these mechanisms may help achieve the efficiency required for small autonomous drones.

At the core of this paradigm are spiking neural networks (SNNs), which differ from conventional artificial neural networks by encoding information through discrete spikes rather than continuous activations (Maass [13]). This asynchronous, event-driven computation is particularly well suited to the resource constraints of small drones, enabling efficient processing of sensory data and significant reductions in energy consumption. In contrast to traditional AI models that update continuously at fixed intervals, SNNs process information only when significant changes occur, making them ideal for low-latency, real-time control.

However, despite these advantages practical deployment remains a considerable challenge. One of the key difficulties lies in the training of SNNs. Standard artificial neural networks – including multilayer perceptrons (MLPs), Recurrent Neural Networks (RNNs) and even the transformers that support Large Language Models (LLMs) such as ChatGPT – mostly rely on gradient-descent methods (such as backpropagation, or backpropagation-through-time (BPTT) (Werbos [14]) for networks with temporal dependencies) for training. Although their activation functions may exhibit non-smooth behavior at certain points, they still enable a stable gradient flow that facilitates effective optimization. In contrast, the spike function in an SNN (often modeled as a Heaviside step function) has a zero gradient everywhere except at its threshold, complicating the application of traditional gradient-based techniques. To address this issue, researchers have developed

[2]Bitcraze: `www.bitcraze.io/`

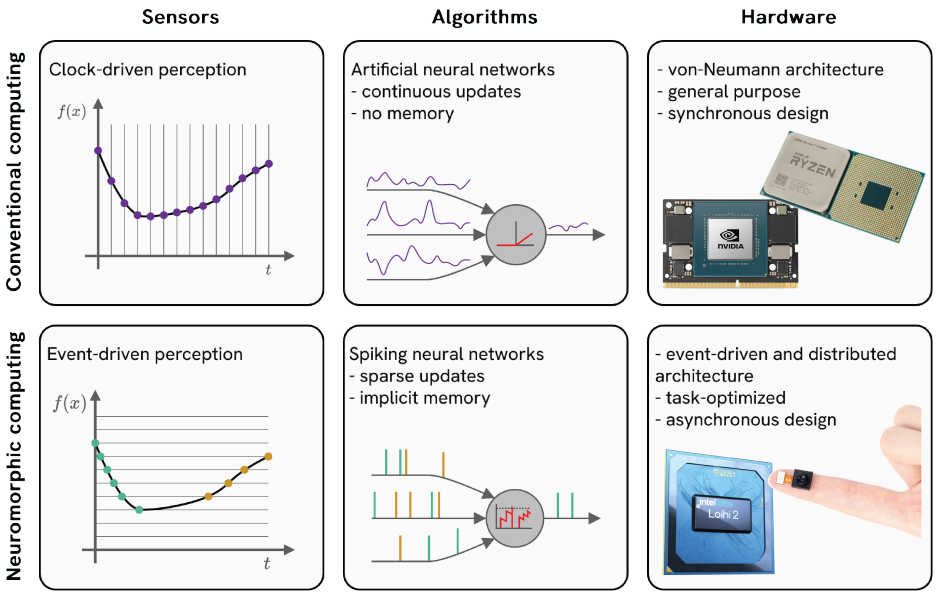


Figure 1.2: Comparison between neuromorphic computing (*bottom*) and conventional computing (*top*). The latter operate using continuous floating point values, while the former use binary spikes to communicate. In this thesis, we have explored all three components (*sensors, algorithms & hardware*) of neuromorphic computing for drones.

alternative strategies, such as surrogate gradient methods (Zenke *et al.* [15]; Zenke *et al.* [16]) and biologically plausible local learning rules (e.g. spike-timing-dependent plasticity, STDP; Caporale *et al.* [17], Diehl *et al.* [18]). Nonetheless, training SNNs remains a difficult task (Tavanaei *et al.* [19]). Moreover, the spike-based encoding of information poses a challenge for regression tasks in particular. The majority of recent work on SNNs is focused on solving slow, static classification tasks, often benchmarking performance on image datasets like MNIST, CIFAR-10 or ImageNet (Nunes *et al.* [20]). However, in robotic applications, these networks must handle dynamic, continuous regression tasks, such as real-time perception and control for flying drones, as explored in this thesis. Conventional neural networks represent continuous values directly as *floating point values*, whereas SNNs must translate this information into discrete events. Common encoding schemes – such as rate coding, where information is represented by the frequency of spikes, temporal coding, which leverages the precise timing of spikes and position coding that uses specific (groups of) neurons to represent a value – each come with

trade-offs in precision, latency, and energy efficiency. The choice of encoding method can significantly impact the network's ability to accurately perform regression, adding another layer of complexity to the design and training of SNNs (Schuman *et al.* [21]). Consequently, bridging the gap between these promising theoretical models and their robust, scalable deployment in real-world systems remains a central challenge.

1.2.1. Neuromorphic Hardware: Current Landscape and Challenges

To fully harness the potential of neuromorphic computing, specialized hardware is required to implement spiking computation efficiently. Conventional CPUs and GPUs excel in dense, numerical processing but are not designed for the sparse, event-driven nature of SNNs. Recent advancements in neuromorphic hardware have sought to address this limitation, with several architectures emerging as key contenders for real-time robotic control, such as 1. Intel's Loihi: One of the most advanced neuromorphic processors, Loihi incorporates 128 neuromorphic cores, enabling low-power, on-chip learning (Davies *et al.* [12]). It has demonstrated promise in real-time robotics but remains limited in its direct integration into flight control systems due to architectural constraints and software compatibility issues. The chip was recently succeeded by the Loihi 2 (Orchard *et al.* [22]), which provides up to 10 times more neurons per chip, enhanced programmability, and improved energy efficiency and speed, enabling even more complex and efficient spiking neural network models, 2. SpiNNaker (Spiking Neural Network Architecture): Designed for large-scale biological neural simulations, SpiNNaker employs a massively parallel architecture capable of modeling millions of spiking neurons (Furber *et al.* [23]). However, its power consumption and hardware footprint make it less suitable for embedded aerial platforms, 3. BrainScaleS: This mixed-signal neuromorphic system leverages analog computation to efficiently simulate spiking activity (Schemmel *et al.* [24]). While it provides a biologically realistic model of neuromorphic processing, its focus remains primarily on neuroscience research rather than real-time embedded control. 4. Synsense's Speck [25]: This system-on-chip (SoC) combines an event-based vision sensor with a processor that can run convolutional spiking neural networks. However, only simple Integrate-and-Fire (IF) neurons are supported. From these, only the Kapoho Bay version of Intel's Loihi is a chip in a USB form factor that can be easily embedded onboard a flying drone. Beyond computational *processors*, neuromorphic *sensing* technologies

are also advancing the field. Event-based vision sensors, such as dynamic vision sensors (DVS), offer a promising alternative to conventional cameras by operating asynchronously – detecting only changes in brightness rather than capturing full image frames at fixed intervals (Gallego *et al.* [26]). This reduces computational overhead and aligns well with neuromorphic principles, making event-based sensing particularly valuable for high-speed drone navigation and control.

1.2.2. Challenges in Neuromorphic Flight Control

However, several fundamental challenges must be addressed before widespread adoption of neuromorphic systems on robots can be realized. One of the most significant obstacles is the "reality gap" – the discrepancy between simulations and real-world deployment, where sensor noise, aerodynamic disturbances, and environmental uncertainty introduce additional complexities (Muratore *et al.* [27]). Most existing state-of-the-art neuromorphic control models for drones are either demonstrated in simulation (Qiu *et al.* [28]) or rely on simplifications of the model that ensure stability (such as the seesaw structure in Stagsted *et al.* [29]). Furthermore, integrating neuromorphic processors and controllers into existing robotic architectures presents additional difficulties. Conventional flight controllers operate using deterministic, time-stepped algorithms, whereas neuromorphic systems function asynchronously and sparse. Achieving seamless communication between neuromorphic estimators, control algorithms, and low-level actuation systems remains an open problem. Additionally, the training and optimization of SNNs for drone control remain largely underexplored, particularly in tasks that require integrating sensor data across multiple timescales. High-frequency inputs, such as IMU-based attitude estimation, demand rapid, fine-grained processing, while lower-frequency tasks, like path planning, rely on more abstract, long-term decision-making. Effectively training SNNs to handle both remains an open challenge.

1.2.3. Future Directions for Neuromorphic Flight Control

A particularly promising avenue is the development of fully integrated neuromorphic autopilots, where decision making, navigation, state estimation and control are combined into a single (modular) spiking network, eliminating the need for traditional microprocessors. Preliminary research has demonstrated that event-based neuromorphic perception can be successfully applied to parts of the control loop of a drone. Vitale *et al.* [30] have shown that event cameras

can be directly connected to a neuromorphic chip to perform horizon tracking on a bi-rotor with incredible speeds. Its application to low-level closed-loop control onboard a flying robot remains an open research problem.

As neuromorphic technology and deep learning research progress, they may enable drones to process sensory data more efficiently while also leveraging advanced learning mechanisms to achieve responsiveness and adaptability akin to biological intelligence.

1.3. Problem Statement and Research Questions

This thesis focuses on the application of neuromorphic systems to quadrotors. As was described in the sections above, drone autonomy could benefit greatly from having fully integrated neuromorphic solutions that act as autopilots. So far in literature, there is little research towards using SNNs for regression tasks, control and estimation in particular, and most was performed in simulation. Therefore, our aim is to perform low-level control onboard a flying drone, which culminates in the following research goal:

> **Research Goal**
>
> Design and train neuromorphic algorithms that perform low-level attitude control

Low-level attitude control, as tackled in this thesis, consists of the commands sent to the actuators that must simultaneously stabilize the drone and adhere to higher-level attitude commands. These attitude commands might for instance be produced by a navigation task, an obstacle avoidance algorithm kicking in or simply commands send by a pilot via a remote. Before being able to control, it is essential to have an accurate estimate of one's *state* - those variables that uniquely describe the future of a dynamical system given past behavior and future inputs (Willems [31]). The most important states for flight control are arguably those related to its attitude, how it is oriented in space. Due to the effects of gravity, control of a drone becomes highly non-linear at larger angles. In a fully neural pipeline, an accurate representation of the current state and an internal model of how it will evolve are thus important.

The first research question investigates how we can achieve this using neuromorphic systems:

> **Research Question 1**
>
> How can we develop and train a neuromorphic system to estimate the attitude of a flying drone?

We will look at this challenge from two perspectives; The first uses spiking neurons to determine its attitude from Inertial-Measurement-Unit sensor data, already exploring how to train SNNs using supervised learning for regression tasks. The second focuses on neuromorphic hardware, especially event-based cameras and how they can be used to determine a quadrotor's attitude while in flight.

After state estimation, we will shift our attention to the isolated case of control. We assume full knowledge of the drone's state and only look at how spiking neurons can be employed to perform control. Attitude control poses some significant issues, it has to deal with derivatives that contain very high frequency information while at the same time dealing with integration, which has very low frequency information. For supervised learning using BPTT, these are already difficult tasks with regular recurrent networks, let alone SNNs.

> **Research Question 2**
>
> How can we design a network of spiking neurons to perform attitude control of a flying drone?

In the scope of this question, *design* means both manually constructing and learning controllers. The benefit of manually tweaking the parameters and connections is that the potential advantages of neuromorphic systems are obtained, while retaining the predictability and robustness of current techniques. Current work on this topic lacks scalability and real-life demonstration. On the other hand, reliably training controllers offers much flexibility and might outperform state-of-the-art controllers (see Song *et al.* [32], for example). As already mentioned, however, training SNNs is difficult.

Finally, we address the integration challenge:

> **Research Question 3**
>
> How can we create an SNN to perform full attitude estimation and control onboard?

This final question investigates how state estimation and control can be merged into a single neuromorphic network, thereby realizing a fully integrated low-level control pipeline.

1.4. Outline

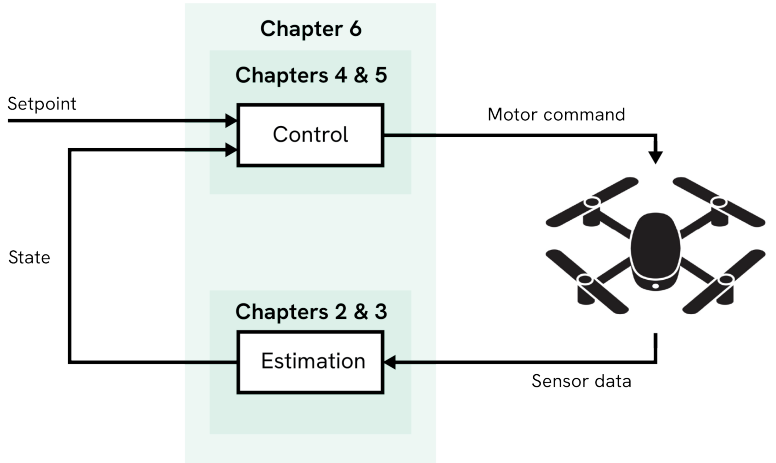


Figure 1.3: Outline of the thesis, with chapters 2 & 3 diving into Research Question 1, chapters 4 & 5 into Research Question 2, and chapter 6 answering Research Question 3 by bringing everything together.

The remainder of this thesis is organized as follows. Chapter 2 explores learning SNNs to perform attitude estimation from IMU data on a drone. This directly delves into Research Question 1. We successfully demonstrate that recurrent SNNs can estimate the attitude of a flying drone, learning from a limited amount of data. The chapter investigates how these networks perform this task and we also show that they can compete with state-of-the-art RNNs in regression tasks. In Chapter 3 we look at using neuromorphic sensors to perform attitude estimation, again looking at Research Question 1. We use an event-based camera to estimate the attitude and angular velocity of a drone

during flight, and demonstrate it in the control loop of a drone with the entire pipeline running onboard. Chapter 4 and 5 focus on Research Question 2, both looking at different methods to perform control with SNNs. The former investigates manually tuned networks that can perform PID control, while the latter allows for training a network that behaves as a PID. A novel trainable adaptation method is described in Chapter 5, where the threshold of a neuron is adapted based on incoming signals. This way, a group of neurons was able to accurately integrate errors over time. Also, Chapter 5 shows some initial investigations in how neurons might perform differentiation. The final Research Question is then tackled in Chapter 6, where we demonstrate a fully spiking network that combines state estimation from IMU inputs with control. We show that fixing certain parameters in the supervised learning pipeline is essential when learning tasks like integrating errors. The performance is demonstrated in real onboard flight on a tiny Crazyflie, attaining high performance with only a limited number of neurons. To conclude, Chapter 7 answers the Research Questions that were posed in the introduction and discusses its implications for future work.

References

[1] S. A. H. Mohsan, N. Q. H. Othman, Y. Li, M. H. Alsharif and M. A. Khan. 'Unmanned aerial vehicles (UAVs): Practical aspects, applications, open challenges, security issues, and future trends'. In: *Intelligent Service Robotics* 16.1 (2023), pp. 109–137.

[2] B. S. Faiçal, H. Freitas, P. H. Gomes, L. Y. Mano, G. Pessin, A. C. de Carvalho, B. Krishnamachari and J. Ueyama. 'An adaptive approach for UAV-based pesticide spraying in dynamic environments'. In: *Computers and Electronics in Agriculture* 138 (2017), pp. 210–223.

[3] M. Lyu, Y. Zhao, C. Huang and H. Huang. 'Unmanned aerial vehicles for search and rescue: A survey'. In: *Remote Sensing* 15.13 (2023), p. 3266.

[4] D. Floreano and R. J. Wood. 'Science, technology and the future of small autonomous drones'. In: *Nature* 521.7553 (2015), pp. 460–466.

[5] G. de Croon. 'Flapping wing drones show off their skills'. In: *Science Robotics* 5.44 (2020), eabd0233.

[6] E. W. Hawkes and D. Lentink. 'Fruit fly scale robots can hover longer with flapping wings than with spinning wings'. In: *Journal of the Royal Society Interface* 13.123 (2016), p. 20160730.

[7] M. Karásek, F. T. Muijres, C. De Wagter, B. D. Remes and G. C. De Croon. 'A tailless aerial robotic flapper reveals that flies use torque coupling in rapid banked turns'. In: *Science* 361.6407 (2018), pp. 1089–1094.

[8] Y. Zhou, B. Rao and W. Wang. 'UAV swarm intelligence: Recent advances and future trends'. In: *Ieee Access* 8 (2020), pp. 183856–183878.

[9] K. McGuire, C. De Wagter, K. Tuyls, H. Kappen and G. de Croon. 'Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment'. In: *Science Robotics* 4.35 (2019), eaaw9710.

[10] B. P. Duisterhof, S. Li, J. Burgués, V. J. Reddi and G. C. De Croon. 'Sniffy bug: A fully autonomous swarm of gas-seeking nano quadcopters in cluttered environments'. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 9099–9106.

[11] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose and J. S. Plank. 'A survey of neuromorphic computing and neural networks in hardware'. In: *arXiv preprint arXiv:1705.06963* (2017).

[12] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.* 'Loihi: A neuromorphic manycore processor with on-chip learning'. In: *Ieee Micro* 38.1 (2018), pp. 82–99.

[13] W. Maass. 'Networks of spiking neurons: the third generation of neural network models'. In: *Neural Networks* 10.9 (1997), pp. 1659–1671.

[14] P. J. Werbos. 'Backpropagation through time: what it does and how to do it'. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[15] F. Zenke and S. Ganguli. 'SuperSpike: Supervised learning in multilayer spiking neural networks'. In: *Neural Computation* 30 (6 June 2018), pp. 1514–1541. ISSN: 1530888X. DOI: 10.1162/neco_a_01086.

[16] F. Zenke and T. P. Vogels. 'The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks'. In: *Neural Computation* 33.4 (2021), pp. 899–925.

[17] N. Caporale and Y. Dan. 'Spike timing–dependent plasticity: a Hebbian learning rule'. In: *Annu. Rev. Neurosci.* 31.1 (2008), pp. 25–46.

[18] P. U. Diehl and M. Cook. 'Unsupervised learning of digit recognition using spike-timing-dependent plasticity'. In: *Frontiers in computational neuroscience* 9 (2015), p. 99.

[19] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier and A. Maida. 'Deep learning in spiking neural networks'. In: *Neural networks* 111 (2019), pp. 47–63.

[20] J. D. Nunes, M. Carvalho, D. Carneiro and J. S. Cardoso. 'Spiking neural networks: A survey'. In: *IEEE access* 10 (2022), pp. 60738–60764.

[21] C. Schuman, C. Rizzo, J. McDonald-Carmack, N. Skuda and J. Plank. 'Evaluating encoding and decoding approaches for spiking neuromorphic systems'. In: *Proceedings of the international conference on neuromorphic systems 2022*. 2022, pp. 1–9.

[22] G. Orchard, E. P. Frady, D. B. D. Rubin, S. Sanborn, S. B. Shrestha, F. T. Sommer and M. Davies. 'Efficient neuromorphic signal processing with loihi 2'. In: *2021 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE. 2021, pp. 254–259.

[23] S. B. Furber, F. Galluppi, S. Temple and L. A. Plana. 'The spinnaker project'. In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665.

[24] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier and S. Millner. 'A wafer-scale neuromorphic hardware system for large-scale neural modeling'. In: *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2010, pp. 1947–1950.

[25] O. Richter, Y. Xing, M. De Marchi, C. Nielsen, M. Katsimpris, R. Cattaneo, Y. Ren, Y. Hu, Q. Liu, S. Sheik *et al.* 'Speck: A smart event-based vision sensor with a low latency 327k neuron convolutional neuronal network processing pipeline'. In: *arXiv preprint arXiv:2304.06793* (2023).

[26] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis *et al.* 'Event-based vision: A survey'. In: *IEEE transactions on pattern analysis and machine intelligence* 44.1 (2020), pp. 154–180.

[27] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger and J. Peters. 'Robot learning from randomized simulations: A review'. In: *Frontiers in Robotics and AI* 9 (2022), p. 799893.

[28] H. Qiu, M. Garratt, D. Howard and S. Anavatti. 'Evolving spiking neurocontrollers for UAVs'. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. Canberra, ACT, Australia: IEEE, 2020, pp. 1928–1935. DOI: 10.1109/SSCI47803.2020.9308275.

[29] R. Stagsted, A. Vitale, J. Binz, A. Renner, L. B. Larsen and Y. Sandamirskaya. 'Towards neuromorphic control: A spiking neural network based PID controller for UAV'. In: *Robotics: Science and Systems 2020, Virtual Conference*. RSS. Robotics: Science and Systems, June 2020. DOI: 10.15607/rss.2020.xvi.074.

[30] A. Vitale, A. Renner, C. Nauer, D. Scaramuzza and Y. Sandamirskaya. 'Event-driven vision and control for UAVs on a neuromorphic chip'. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. Xi'an, China: IEEE, 2021, pp. 103–109. DOI: 10.1109/ICRA48506.2021.9560881.

[31] J. C. Willems. 'Paradigms and puzzles in the theory of dynamical systems'. In: *IEEE Transactions on automatic control* 36.3 (1991), pp. 259–294.

[32] Y. Song, A. Romero, M. Müller, V. Koltun and D. Scaramuzza. 'Reaching the limit in autonomous racing: Optimal control versus reinforcement learning'. In: *Science Robotics* 8.82 (2023), eadg1462. DOI: 10.1126/scirobotics.adg1462. URL: https://www.science.org/doi/abs/10.1126/scirobotics.adg1462.

2

Attitude estimation using spiking networks

In this work, we propose a spiking neural network (SNN) capable of estimating the pitch and roll angles of a quadrotor in highly dynamic movements from 6-degree of freedom Inertial Measurement Unit (IMU) data. With only 150 neurons and a limited training dataset obtained using a quadrotor in a real world setup, the network shows competitive results as compared to state-of-the-art, non-neuromorphic attitude estimators.

The proposed architecture was successfully tested on the Loihi neuromorphic processor on-board a quadrotor to estimate the attitude when flying. Our results show the robustness of neuromorphic attitude estimation and pave the way towards energy-efficient, fully autonomous control of quadrotors with dedicated neuromorphic computing systems.

Parts of this chapter have been published in Neuromorphic Computing and Engineering [1]

2.1. Introduction

Over the last two decades, efforts have been made to combine the fields of Artificial Intelligence (AI) and Robotics with outstanding results [2]. Algorithms making use of AI techniques, such as deep neural networks, have been proposed to achieve state estimation [3], object manipulation [4], localization [5] and control [6–8]. For instance in [9], a quadrotor learns to fly by applying Reinforcement Learning (RL) to a densely connected neural network. Aerial vehicles are critical embedded systems, the constraints of which (e.g., size and weight, battery autonomy, sensors, computing resources) hamper the development and the performance of fully autonomous and robust on-board control. In particular, Micro Air Vehicles (MAVs) are a class of aerial vehicles that could strongly benefit from AI-powered solutions to handle their highly non-linear dynamics, and allow for online adaptations to unpredictable changes occurring in the real world (e.g., gusts of wind, sensor damage, communication failure, etc.). Many of the complex tasks future MAVs will have to perform will be powered by AI. One can think of deep neural networks that have to estimate optical flow [10] and recognize objects [11], which can be used for vision-based autonomous navigation in urban areas to deliver packages.

However, MAVs are now locked out from using large-scale neural networks because of the great amount of energy required, as well as disproportionate need for computing resources that only Graphics Processing Units (GPUs) can offer. Additionally, standard Von Neumann architectures suffer from a relatively high latency that restrict their use in extreme conditions such as drone racing and aggressive flight maneuvers.

Alternatively to traditional Artificial Neural Networks (ANNs) where the information is processed in a synchronous manner, Spiking Neural Networks (SNNs) could represent the choice solution to bridge the gap between AI and resource-restricted Robotics. In contrast to ANNs, SNNs encode the information not by the intensity of the signal, but by a series of binary events, also called spikes, and the relative time between them. Inspired by their biological counterpart, spiking neurons accumulate incoming synaptic currents over time and fire whenever their membrane potential exceeds a certain threshold. While a wide range of neuron models have been proposed, the most commonly used are the Integrate-and-Fire (IF) and Leaky-Integrate-and-Fire (LIF) [12]. The simpler binary signals and sparse firing of SNNs hold the promise of orders of magnitude more energy-efficient processing than ANNs [13].

The move towards SNNs requires a complete shift in the coding and processing

of information that is not optimized for Von Neumann architectures. In this regard, neuromorphic sensors and processors have been designed, arousing the enthusiasm of roboticists to embed these new technologies onboard robots. Examples of neuromorphic processors include HICANN [14], NeuroGrid [15], IBM's TrueNorth [16], APT's SpiNNaker [17] which is part of the Human Brain Project [18], and Intel's Loihi [19]. In terms of neuromorphic sensing, most efforts have been put to develop the neuromorphic equivalent to standard CMOS cameras, namely, event-based cameras [20]. Neuromorphic tactile sensors have also been proposed [21].

Tackling computationally expensive vision or navigation tasks with neuromorphic sensing and processing will bring large energy and speed benefits. However, to optimally reap the benefits of neuromorphic algorithms and hardware, an end-to-end, fully neuromorphic solution needs to be designed. Then a single neuromorphic processor could suffice. In literature, some of the steps towards such a fully neuromorphic pipeline have been demonstrated. In Vitale *et al.* [22], the authors introduced a neuromorphic PD controller that outperforms state-of-the-art controllers in high-speed control thanks to the synergy between the high update rates of the neuromorphic chip and the event-camera. Also, autonomous thrust control of a flying quadrotor was achieved by an SNN evolved in simulation, using optic flow to enact a constant-divergence landing [23]. In [24], the authors show that SNNs are capable of solving planning tasks, such as avoiding obstacles with a robotic arm.

This fully neuromorphic pipeline will also have to include "lower-level" tasks, but these are not well studied until now. For example, autonomous control onboard MAVs requires an accurate estimate of the states, such as attitude and global lateral position and velocity, by combining sensor measurements. In this article, we propose a neuromorphic solution for onboard attitude estimation of a MAV using data from an inertial measurement unit (IMU), combining data from a 3-axes accelerometer and a 3-axes gyroscope. We compare the network to a similarly sized and trained traditional Recurrent Neural Network (RNN) and commonly used filters specific to this task. The proposed SNN is trained from limited data obtained with a real MAV and can be employed as part of an autonomous neuromorphic flight-controller pipeline for an MAV. Closest to our work is the study in [25], which trains a traditional ANN with recurrency for attitude estimation. Specifically, in [25] it is shown that a 2-layer Recurrent Neural Network (RNN) can be trained to perform this task with outstanding results on pre-gathered datasets. This network, however, is not spiking and has not been applied in the control loop of MAVs in flight.

MAVs are usually equipped with an IMU and use the combination of the angular velocities and the linear accelerations to estimate the current attitude. Angular velocities show the rate of change, but induce integration errors over longer time-windows while the linear acceleration shows the gravity vector over longer stretches of time. The output pitch and roll estimates are necessary for the drone to be able to control the position in the x-y plane by performing attitude control.

Our contributions are threefold. First, we propose an SNN architecture to perform state estimation for dynamic systems such as quadrotors with limited data obtained with a physical quadrotor and ground-truth. Then, we demonstrate that the proposed neuromorphic state estimator exhibits competitive performance when compared to widely used, non-neuromorphic solutions (i.e., Madgwick filter, Mahony filter, and complementary filter) and to a traditional RNN. Lastly, we successfully test our solution onboard a quadrotor equipped with the Loihi neuromorphic chip, thus paving the way for a fully neuromorphic control-loop for quadrotors.

2.2. Methods

2.2.1. Spiking network definition

This section introduces the attitude estimation spiking neural network, called Att-SNN. The different components that make up the network are shown in Figure 2.1.

Spiking neuron model

In this work, we use the Leaky-Integrate-and-Fire (LIF) neuron as the core of our SNN. Widely used in the literature, the LIF model is available in most SNN simulators a neuromorphic hardware, including the Loihi used for this study [19]. The discrete-time difference equations governing the LIF neuron are given as follows:

$$v_i(t+1) = \tau_i^{\text{mem}} v_i(t) + i_i(t) \tag{2.1}$$

$$i_i(t+1) = \tau_i^{\text{syn}} i_i(t) + \sum w_{ij} s_j(t) \tag{2.2}$$

where $v_i(t)$ is the membrane potential at time t, $\tau_i^{\text{mem}} \in [0, 1]$ and $\tau_i^{\text{syn}} \in [0, 1]$ the membrane and synaptic time constants, $i(t)$ the synaptic current at time t, w_{ij} the synaptic weight between neurons i and j, and s_j a binary value representing either a spike or no spike coming from the pre-synaptic neuron j. To determine whether a neuron emits a spike, the membrane potential is reduced with the neurons firing threshold θ_i^{thr} and passed through the Heaviside step-function to determine the

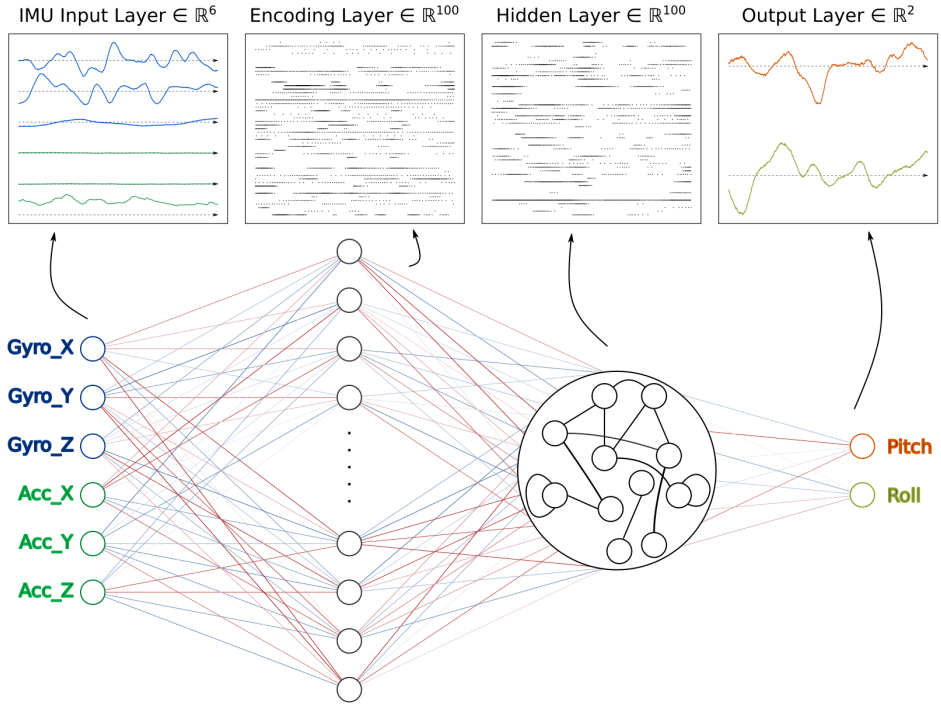


Figure 2.1: Topology of the attitude estimation network Att-SNN showing signals transported between layers. In the encoding layer, the normalized IMU data is transformed into spikes. The next layer is a fully connected recurrent layer that sends spikes to the Leaky-Integrator (LI) output layer, converting spikes back to attitude estimates.

output of the neuron:

$$s_i(t) = H(v_i(t) - \theta_i^{\text{thr}}) = \begin{cases} 0, & v_i(t) - \theta_i^{\text{thr}} \leq 0 \\ 1, & v_i(t) - \theta_i^{\text{thr}} > 0 \end{cases} \tag{2.3}$$

When the Heaviside-function resolves to 1 and the neuron emits a spike, the membrane potential $v_i(t)$ is reset to zero.

Data encoding

Standard, off-the-shelf IMUs are not neuromorphic; the output data is formatted as floating point values and streamed synchronously. As a result, the measured angular rates and linear accelerations must be translated into spikes so that they can be processed by the SNN.

Data encoding is a complex task. Spike coding algorithms can be divided into

three categories. Population coding uses set of distinct neurons to encode (or decode) the signal by emitting a spike whenever the input signal falls within the range distribution of one (or several) neuron. It has been successfully applied in [23, 26]. Temporal coding algorithms encode the information with high timing precision, by emitting a spike whenever the variation of the input signal exceeds a threshold. Rate coding, which was used in [24], encodes the information into the the firing frequency of a population of neurons.

In this work, the floating point values returned by the 6-DOF IMU are encoded into spikes by means of a spiking layer densely connected to the 6 outputs of the IMU sensor (Fig. 2.1). The synaptic weights, as well as the dynamics of the neurons in this encoding layer were learned offline. Both the time-constants were trained for each neuron seperately, resulting in $2N$ parameters, with N the number of neurons in the encoding layer. Before passing the IMU data to the network, a normalization process is applied to ensure faster learning and convergence during training. This normalization is achieved through min-max, mapping the input data to a range of $[-1, 1]$, according to the formula hereafter. The values of x_{min} and x_{max} were chosen empirically to match the full range of possible inputs.

$$x_{\text{norm}}(t) = 2 \cdot \frac{x(t) - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} - 1 \tag{2.4}$$

Decoding the global attitude

Decoding of the spiking activity to an estimate of the attitude in radians is performed via a non-spiking decoding layer (Fig. 2.1), composed of two Leaky-Integrate (LI) neurons. In this case, the membrane potential $v_i(t)$ in Equation 2.1 of the neuron is directly used as the output of the layer, providing a stateful decimal value representing the current pitch or roll attitude. In contrast to the input-data, no further normalization is required for the attitude values. The pitch and roll angles in the training data, expressed in radians, are already distributed around a mean of zero with standard deviation of 0.25. Theoretically, the limits go from $-\pi$ to π, but since we are targeting a quadrotor in a normal flight regime around hover, these values will not be reached. Lastly, since the pitch and roll axes of the MAV are symmetric (this is true in the case of a standard, symmetric quadrotor), the neuron parameters of the decoding layer are set equal during training. This results in training two parameters for this layer.

2.2.2. Training

Training setup

The proposed Att-SNN was trained both in simulation, using datasets created with the RotorS simulator [27] (Fig. 2.2), and with data collected with a quadrotor flying in the real world[1] (Fig. 2.2). The quadrotor is equipped with a Pixhawk 4 Mini flight controller, which combines the measurement of two separate IMU's for redundancy. The IMU data are logged at 200Hz. For these experiments, the ground truth was provided by a multi-camera motion capture system (OptiTrack). This system provides sub-degree accuracy attitude estimates at the same rate as the combined IMU measurements from the quadrotor. An overview of the distribution of the IMU and OptiTrack data collected both in simulation and in the real world is provided in Figure 2.3. In total, 35 datasets of 100 seconds were gathered in simulation and 11 datasets of 100 seconds in our real-world tests. This amounts to a total of 77 minutes of flight time. Both the simulation and real-world data sets have been gathered in a way as to represent roll and pitch angles between hover and swift flight, reaching relatively large angles up to 45 degrees. This covers a normal flight regime. An important challenge in IMU-based attitude estimation is that the gyros and accelerometers have biases that can change over time. For instance, accelerometers usually suffer from a turn-on bias which shows as a constant offset of the measured acceleration. This bias is especially visible in the x-axis of the PX4 accelerometer. It is also worth noting that there is also a difference in the range of values in the $gyro_z$, which shows that the quadrotor in simulation was rotating more around the z-axis than in the real-world experiments. Since this rotation is only affecting the yaw angles directly, the influence on the pitch and roll estimates is limited. The datasets were split up in 70% train and 20% validation, and 10% test. The training datasets were split into sequences of 10 seconds containing 2000 time steps. Furthermore, the simulated data was augmented by adjusting the accelerometer turn-on bias and both the accelerometer and gyroscope noise densities to reduce the reality gap. The noise characteristics were determined using the steady-state error of the real world data set.

The Att-SNN was implemented using the Norse [28] python library, based on PyTorch. The Adam optimizer [29] was used with a learning rate of 0.005, combined with the k-step ahead, 1 step back Lookahead [30] optimizer to speed up learning. For the Lookahead optimizer, the value of α was set at 0.5 and k at 6.

[1] The dataset can be found in `https://doi.org/10.4121/20464830.v1`

All code to reproduce the results can be found in [2].

Since the Heaviside function used in the neuron dynamics (Eq. 2.1) is non-differentiable, a surrogate-gradient (SG) was chosen to enable backpropagation through time (BPTT). A summary of SGs can be found in [31]. In this study, we used the SuperSpike [32] with a width of 20 as it is a suitable option for supervised BPTT and is robust to changes in the input paradigm as is the case with our current-based input [33].

The error between the output and the target was characterized by the Mean-Squared-Error (MSE) loss function where both the pitch- and roll-error were weighted evenly. The MSE for a sequence S is calculated as in Equation 2.5, with $\hat{\theta}_k$ and θ_k^{gt} the estimated and ground-truth pitch angles at timestep k, and $\hat{\phi}_k$ and ϕ_k^{gt} the estimated and ground-truth roll angles.

$$\text{MSE}_S = \sum_{k=0}^{N=\text{len}(S)} \frac{(\hat{\theta}_k - \theta_k^{\text{gt}})^2 + (\hat{\phi}_k - \phi_k^{\text{gt}})^2}{2} \tag{2.5}$$

The Att-SNN is trained with 100 neurons in the encoding layer and 100 in the hidden layer. The thresholds were fixed at 0.5 for all neurons. This results in 600 weights and 200 neuron parameters for the encoding layer, 10.000 + 10.000 weights and 200 neuron parameters for the hidden layer and 200 weights and 2 neuron parameters for the output layer. In total, this adds up to training of 20.800 weights and 402 neuron parameters. Every epoch, 15 batches of 40 sequences are randomly selected from all training data (including simulated and real-world data) for a training iteration, and afterwards the error is calculated for all validation sets. The training is stopped by a criterion based on a moving average of the error on the validation dataset over the last 20 epochs. If the moving-average of the error is higher than 110% of the lowest average validation loss so far, training is aborted. Besides, if the lowest validation loss so far did not change for at least 50 epochs, training is also aborted.

2.2.3. Implementation on neuromorphic hardware

To fully demonstrate the potential of SNNs for state estimation in MAV applications, the proposed Att-SNN architecture has been implemented on Intel's Loihi processor [19]. The constraints of SNN design imposed by the Loihi requires to adapt the SNN parameters, such as the synaptic weights and the neurons' parameters. A naive solution to this would be to quantize the parameters after training, but this would inevitably result in a loss in accuracy, and in the long run

[2] Code is available in `https://github.com/tudelft/neuromorphic_attitude_estimation`

Figure 2.2: Quadrotors used in this research. The AscTec Hummingbird flying in open-source RotorS quadrotor simulator [27], used to gather supplementary simulation data for training on the *left*. The quadrotor used to gather real-world data and for the evaluation of our SNN on the neuromorphic processor Loihi on the *right*.

could affect the overall performance of the state estimation (and control). Alternatively, we have included the quantization in the training process by replacing the full-resolution weights with their quantized equivalents before the forward pass while propagating the gradients. By doing so, we ensure that the network converges to a solution that is fully compatible with the specific features of the Loihi chip. The quantization function is defined by:

$$p_q = \text{round}(p/\Delta q)\Delta q, \quad p_q \in [q_{\text{min}}, q_{\text{max}}] \tag{2.6}$$

with p_q the quantized version of parameter p, Δq the quantization step-size, round$(.)$ rounding of a floating point value to the closest integer and $[q_{\text{min}}, q_{\text{max}}]$ the quantization range. The threshold θ_{thr} was fixed during training, while all other parameters were not. The quantization ranges and step sizes used in this study are:

- synaptic weights w_{ij}, range: $[-1, 1 - \frac{1}{256}]$, step size: $\frac{2}{256}$
- synaptic decay $\tau_{\text{syn},i}$, range: $[0, 1]$, step size: $\frac{1}{4096}$
- membrane decay $\tau_{\text{mem},i}$, range $[0, 1]$, step size: $\frac{1}{4096}$

During implementation on the neuromorphic chip, the parameters can be multiplied by the quantization range, resulting in integers compatible with the constraints set by the neuromorphic processor.

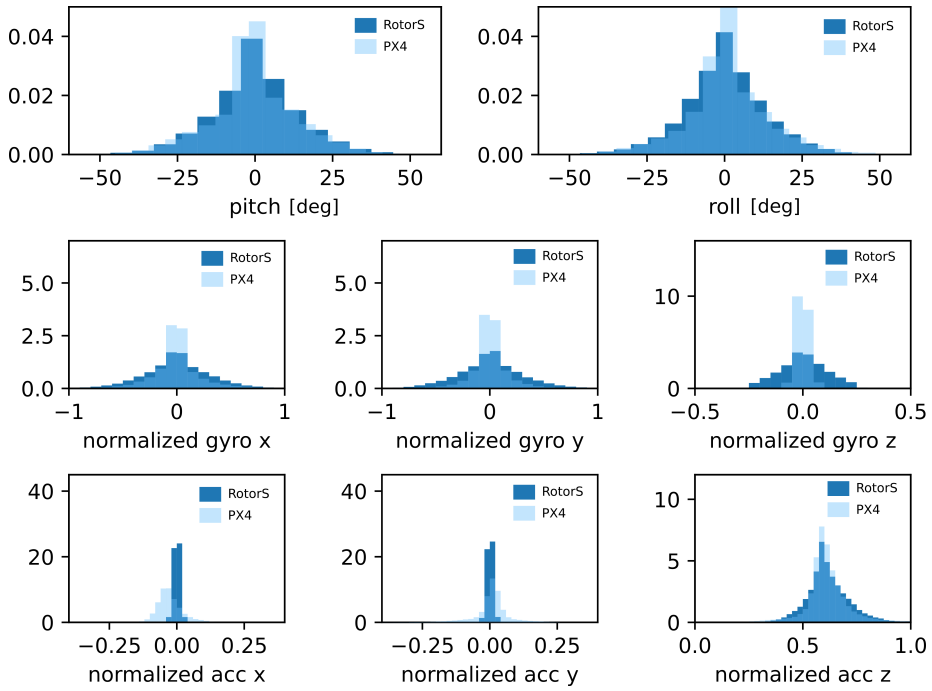


Figure 2.3: Distribution of the input-output data used for training the network. The top two histograms depict the output angle variables. The inputs (lower six histograms) show sensor-readings after applying normalization.

2.2.4. Models for comparison

The performance of the Att-SNN neuromorphic state estimator is compared to the most widely-used non-neuromorphic algorithms: (i) the Madgwick filter [34], (ii) the Mahony filter [35], the Extended Kalman filter (EKF) [36, 37], and (iv) the complementary filter [38]. These filters all adequately estimate the pitch and roll angles of a quadrotor in flight. Since only data from a 6-DOF IMU is used, these two are the only observable states out of a full non-linear model of a quadrotor. For fairness of comparison, we implement a minimalistic EKF that uses the IMU together with a quaternion-based random walk model to estimate pitch and roll. Typically, EKFs are more extensive, as they can estimate more states such as velocity or position, integrate other sensors like GPS, and can employ more detailed motion models using known control inputs. However, such extensions are out of the scope of the current study.

These filters all require some fine-tuning of their parameters with respect to the dynamic motions captured in the datasets, the noise level measured in the IMU,

and a prior knowledge of the initial state. Tuning these parameters by hand is quite common, but this may have resulted in unfair comparison with a neural network that is automatically and thoroughly trained with dedicated algorithms on the training dataset consisting of simulation and PX4 data. As a result, the parameters of all filters were automatically determined for the same datasets using the modified Particle Swarm Optimization (PSO) algorithm as described in [39]. The optimization values were chosen as: $w = 0.8$, $c_1 = 0.15$ and $c_2 = 0.05$ and a total of 100 particles was used. The cost function was defined as the MSE per timestamp, averaged over all training sequences plus a high cost of 10 for parameters below 0 or above 1 to constrain the parameter to the corresponding ranges. Using the PSO, the optimal parameters were estimated for both the simulation and PX4 datasets, assuming that the filters had no knowledge of the initial angle. The impact of this assumption will be further analyzed in Section 2.3.1.

Additionally, we propose a traditional recurrent neural network, called Att-RNN which is composed of gated recurrent units (GRUs). This network has a very similar structure to our proposed Att-SNN, allowing to evaluate the difference in performance caused by the introduction of the LIF model's neural dynamics and spiking.

The complementary filter

The complementary filter is a widely used filter for attitude estimation due to its simplicity. The output of the filter is a weighted average between the angle θ_k^{acc} measured by the gravity vector measured with the accelerometer (when the assumption of weak-acceleration relative to gravity holds), and the previous estimated angle $\hat{\theta}_{k-1}$ propagated with the angular velocity ω_k^{gyr} measured by the gyroscopes. For a single axis this is defined as follows:

$$\hat{\theta}_k = \gamma(\hat{\theta}_{k-1} + \omega_k^{\text{gyr}} \Delta t) + (1 - \gamma)\theta_k^{\text{acc}} \tag{2.7}$$

Where γ is the weighting factor, balancing between the accelerometer that provides a solid baseline on a long time horizon and the gyroscope that is accurate for updating the angle on short time-scale, but suffers from integration errors on a longer scale.

The Mahony filter

Gyroscope measurements from low-cost IMUs can be biased, and the Mahony filter is an extension of the complementary filter that counters this [35]. By adding

an integral term $\hat{b}_k$ to the filter, based on the error e_k between the angle measured by the accelerometer and the predicted angle, the gyroscope biases can be effectively canceled without increasing the computationally cost too much. The filter, in discrete quaternion form, can be written as:

$$\hat{q}_k = \hat{q}_{k-1} + (\frac{1}{2}\hat{q}_{k-1} \otimes \mathbf{p}\{w_k^{\text{gyr}} - \hat{b}_k + k_P e_k\})\Delta t \tag{2.8}$$

$$\hat{b}_k = \hat{b}_{k-1} - k_I e_k \Delta t \tag{2.9}$$

Where $\mathbf{p}\{x\} = [0\ x]^T$ is a pure quaternion that relates to the rotation velocity of the attitude and $p \otimes q$ is the Hamilton product between quaternion p and q. This algorithm can be optimized for certain motions or sensors by adjusting the proportional (k_P) and integral (k_I) gain.

The Madgwick filter

Madgwick [34] defines attitude estimation as a minimization problem, solved with a gradient descent algorithm. While the gyroscope is still used for integrating the angle, the error with respect to measured gravity is represented as a cost function $f(q)$ describing the difference between the angle measured with the accelerometer and the gravity vector rotated to the body-frame. This cost-function is minimized by taking a single gradient descent step. This results in the following update step in discrete quaternion form:

$$\hat{q}_k = \hat{q}_{k-1} + (\frac{1}{2}\hat{q}_{k-1} \otimes \mathbf{p}\{w_k^{\text{gyr}}\} - \beta \frac{\frac{1}{2}\text{Jac}(f(\hat{q}_{k-1}))^T f(\hat{q}_{k-1})}{\|\frac{1}{2}\text{Jac}(f(\hat{q}_{k-1}))^T f(\hat{q}_{k-1})\|})\Delta t \tag{2.10}$$

Where $\text{Jac}(f(\hat{q}_{k-1}))$ is the Jacobian matrix of the cost-function evaluated at $\hat{q}_{k-1}$ and β the optimization parameter. Although this filter can be used for 6-DOF IMU data, it is typically used on 9-DOF IMUs that also have a tri-axial magnetometer. The output quaternions of both the Madgwick and Mahony filters will be transformed to Euler angles for comparison with the other methods.

The Extended Kalman filter

The Extended Kalman Filter (EKF) is one of the most-used fusion algorithms for non-linear systems and can also be applied to attitude estimation with a 6-DOF IMU [40]. It uses a model of the system to predict the state at the next time-step and corrects this prediction with measurements using a prediction of the estimate covariance. This implementation uses the angular rate measured by the gyroscope in the prediction step and the angle measured from the accelerometer measurements in the correction step. Since noise on the dynamic model and

sensor measurements is defined by covariance matrices, the balance between the gyroscope and accelerometer can be optimized by changing these matrices.

A non-neuromorphic neural network

It is not common to use ANNs for attitude estimation, although there is an increasing interest in using ANNs for low-level estimation and control tasks. Here, we also compare our neuromorphic algorithm with a more traditional neural network. As a comparison to the Att-SNN, it was decided to implement a recurrent neural network consisting of two layers with GRUs similar to the one proposed in [25]. However, in our implementation the network has Euler-angles in radians as outputs, instead of quaternions to keep it comparable to our Att-SNN, during training and evaluation. The recurrency and memory in the GRUs allow the network to store and keep track of the states in the network. This is something a regular feedforward ANN would not be able to do, but that is important for integrating and filtering information over time for state estimation. The network has the same number of neurons per layer as the Att-SNN and is trained using the same optimizer and training strategy. The differences lie in (i) the Att-ANN uses GRUs, while the Att-SNN does not, (ii) the neural dynamics, where the Att-ANN sets its state directly based on all feedforward and recurrent connections, and Att-SNN implements the LIF neuron model.

2.3. Results

In the following, we investigate the performance of the proposed Att-SNN. We first compare the accuracy of the Att-SNN with state-of-the-art, non-neuromorphic attitude estimation filters commonly in control of quadrotors. The spike activity of the Att-SNN is then evaluated with respect to the maneuvers performed by the quadrotor to better characterize the dynamic response of the Att-SNN. Lastly, the neuromorphic state estimator is implemented within the control loop to demonstrate the stability and robustness over time while unknown disturbances are applied to the MAV.

2.3.1. Performance analysis

First, we look at the performance of the neural network methods during training, where we compare three variants of the Att-SNN with the Att-RNN (Fig. 2.4). The Att-SNN and the Att-RNN models differ in two aspects. The training of the Att-RNN model results in a more important decrease of the loss function (MSE, Eq. 2.5) than for the neuromorphic model. Moreover, the validation loss of the Att-SNN

model reveals a higher stochasticity (standard deviation of loss minus the moving average of 1.7×10^{-3} vs. 0.9×10^{-3}) than that of the Att-RNN model, which can be explained by the non-differentiability of the Heaviside function used in the LIF model (Eq. 2.3). Both networks suffer to some extent from overfitting, since the loss on the training data is lower than that on validation data. By using data with a wide range of dynamic motions and early-stopping during training, generalization can be improved. The influence of the dataset on the ability of the networks to generalize over data samples never seen during training can be identified by performing a k-fold test. We have split up all PX4 and simulation datasets in 70% train, 20% validation and 10% test data in 5 different folds. We have also included two folds where the network trained on simulation is tested on the PX4 data and vice versa. These are depicted as *sim* and *PX4*. The mean error and standard deviation (SD) (Table 2.1) for these folds on the corresponding test sets show that the model is resilient to the motions observed in the training data, and new motions in the test set. The error on the test dataset for the second fold is even lower than on the training sets. This can be caused by less dynamic maneuvers in the datasets that were kept apart for this specific fold. It also shows that training on a single-source dataset results in extreme generalization errors, indicating the importance of mixed-data.

	Train error		**Test error**	
Fold	**Mean**	**SD**	**Mean**	**SD**
1	2.09	0.32	3.43	0.44
2	2.21	0.50	1.78	0.32
3	1.82	0.25	2.42	0.39
4	2.05	0.38	2.65	0.48
5	2.18	0.51	2.99	0.47
sim	1.97	0.23	4.86	0.80
PX4	2.20	0.41	5.40	0.65

Table 2.1: Results of the k-fold cross-validation showing the mean and standard deviation on the test- and training datasets.

To allow the comparison of performance between all filters, the average angle error between the filter and the ground truth was determined for 40 sequences spanning 50 seconds of real-world flight.

In Figure 2.5, we show the overall results of the two neural-based models (Att-SNN and Att-RNN) along with the four filters (Madgwick, Mahony,

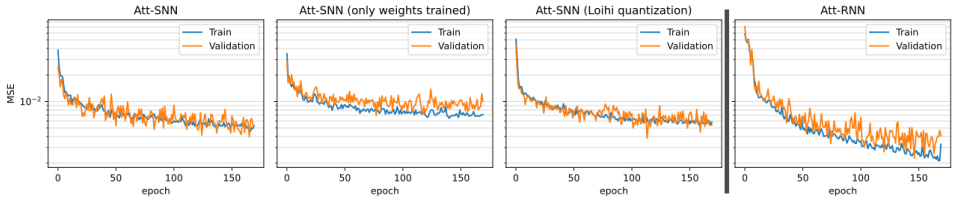


Figure 2.4: Training loss curve of the Att-SNN vs. the similarly trained Att-RNN with gated recurrent units (GRUs) on training and validation data.

Complementary, and EKF) with respect to the training PX4 datasets and the validation PX4 datasets. Overall quantitative results are further detailed in Table 2.2. The performance of the standard filters has been measured both with and without prior knowledge of the initial state. The error distribution of the Att-SNN model shows that the network is able to estimate the attitude of the quadrotor with an error comparable to the common filters. The neural-based methods perform better than the standard filters on the training set, when those filters do not have access to the initial state of the quadrotor (i.e., ^{1} in the table). On the test set, all four filters perform slightly worse as compared to the results obtained with the training set, showing that the test dataset poses certain challenges that are more difficult for the traditional filters to handle. As expected, the performance of the neural based solutions deteriorates slightly as they now face unknown data. This overfitting may be caused by unknown motions or noise characteristics in the test set and could be reduced by increasing the training data. Adding data from different quadrotors that have different dynamic properties and IMUs with different noise characteristics will increase the generalization properties of the network. However, the average error still remains stable, below 2.5° which is in the same range as the errors of the common filters. In Figure 2.6, we further demonstrate the impact of the initial attitude on the response of the filters. Due to the symmetry of the quadrotor along the x and y axes, only the pitch results are presented. Whereas both the Att-SNN and Att-RNN are able to converge to the real angle in less than a second (200 samples), the three conventional filters have not fully converged after 7 seconds. Since the filters all balance between the accuracy on short time-scales given by the gyroscopes and the long time-scale reference of the accelerometer, a large offset at $t = 0$ can result in a long convergence. This balance is forged by the optimization parameters that were found with the PSO, which remain equal over the course of a sequence. These results suggest that the neural networks

have learned a way to perform this balance in an adaptive way, trusting the angle θ_k^{acc} more when they largely contradict the current $\hat{\theta}_k$, but rely less on these estimates when the assumption of weak-acceleration does not hold. An implementation of such an adaptive mechanism for attitude estimation with a 9-dof IMU with an adaptive EKF was shown in [41]. To support this hypothesis, the plain complementary filter is adjusted by an adaptive law, trusting more on θ_k^{acc} if the gyroscopes show low angular velocity. If $\|\omega_k^{\text{gyr}}\| < 0.1$, the complementary gain γ is increased by the difference between the estimated angle $\hat{\theta}_k$ and the angle θ_k^{acc} times a gain k_a chosen as 0.01. The result, as seen in Fig. 2.6, is that the complementary filter recovers faster from an initial offset. Although this does improve the convergence, it is still slower than the neural-based solutions, and is less responsive during aggressive maneuvers. However, it might give some insight in the way the neural-based solutions handle this.

	Training set			**Test set**		
	Median	**Mean**	**SD**	**Median**	**Mean**	**SD**
Att-SNN (this work)[1] [2]	1.90	**1.56**	**0.24**	2.79	2.79	0.37
Att-RNN[2]	**1.53**	1.91	0.28	2.45	2.45	0.35
Madgwick	2.79	2.81	0.36	2.89	2.92	0.41
Madgwick[2]	3.03	3.22	0.83	3.06	3.34	1.11
Mahony	2.56	2.57	0.36	2.53	2.54	**0.33**
Mahony[2]	2.69	2.85	0.74	2.59	2.84	0.89
Complementary	2.42	2.46	0.42	**2.13**	**2.13**	0.35
Complementary[2]	2.60	2.98	1.24	2.36	2.71	1.36
EKF	2.67	2.75	0.52	2.36	2.40	0.44
EKF[2]	2.65	2.71	0.51	2.49	2.64	1.00

Table 2.2: Median, Mean, and Standard Deviation (SD) for the Training and Test sets.

2.3.2. Spiking activity

After training the Att-SNN model, we measured the overall spiking activity over the datasets of both the encoding and the hidden layers to evaluate the sparsity of the network. A histogram of the average spiking activity per neuron is given in Figure 2.7. Results show that, out of all neurons, 27% of neurons in the encoding layer and 10% of neurons in the recurrent layer do not spike at all, and that an

[1] This work.

[2] No knowledge of initial state.

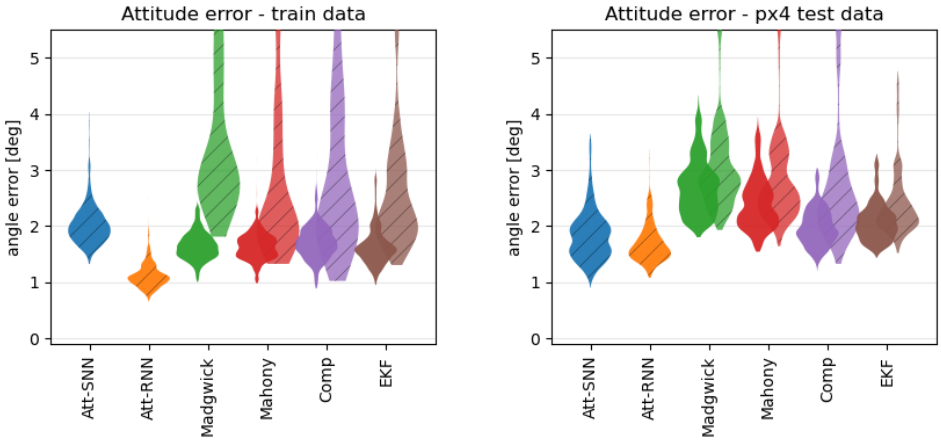


Figure 2.5: Error distribution of the Att-SNN versus other filters for data obtained in flight for training data (*left*) and test data (*right*). The diagonally hatched plots show the error when the initial angle is not known. Both the Att-SNN and Att-RNN always start without knowledge of the initial angle. Results are given in degrees and combine the results for both the pitch and roll angles.

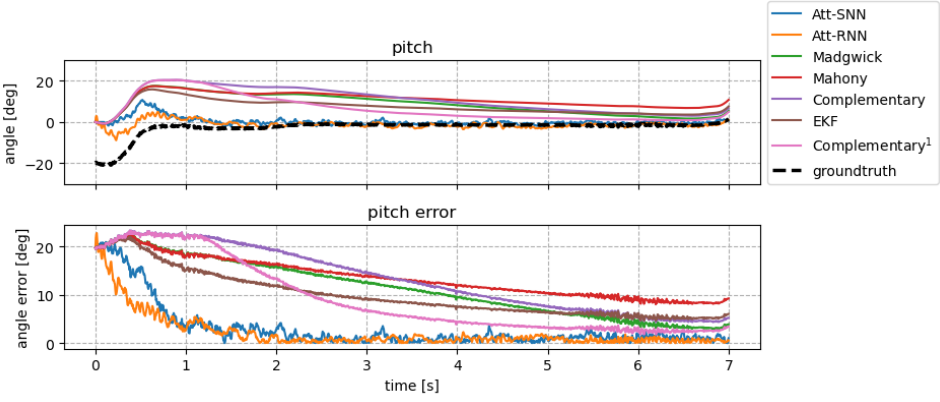


Figure 2.6: Pitch response and error for different filters w.r.t. ground truth for initial offset of ≈20°. In this case, all conventional filters are tested without any information about the initial angle. Real world PX4 data was used for this comparison. The Complementary[1] shows the result of the complementary filter with an adaptation law to allow for quicker convergence.

extra 7% and 8% respectively spike less than 0.5% of the time. To understand if these sparsely spiking neurons have a significant effect, an ablation study has been performed. By pruning all neurons that spike less than **x**% of the time for a subset

of data, the effect on the accuracy for the rest of the data can be established. The results are shown in Figure 2.7. Pruning the neurons that do not exceed an average spiking activity of 0.5% did not modify the performance of the network. This corresponds to a total of approximately 25% of the neurons of the Att-SNN model, which amounts to a reduction of almost half of the synapses while making the network more efficient.

2.3.3. Input dataset manipulation

In order to obtain some preliminary understanding of the functioning of the SNN, we have evaluated the fusion of sensory information by manipulating sensory inputs. In particular, we compared the output of the Att-SNN model during a constant velocity flight where the attitude is kept constant. In Figure 2.8, the response of the network is provided for the pitch angle of the quadrotor obtained in simulation and in the following conditions: (i) the gyroscope is removed, (ii) the accelerometer is removed, (iii) the accelerometer data all point towards gravity in the world frame (resulting in $\theta_k^{\text{acc}} = 0$), and (iv) no manipulation (i.e., the initial Att-SNN). We observe that removing the angular velocity information provided by the gyroscope results in large errors during fast motion, but remains correct when the motion follows a constant angle. This shows that the Att-SNN model uses the accelerometer data for a long time scale estimation of the angle, similar to how the common filters use it. When the accelerometer data is replaced by the gravity vector in the world frame, effectively measuring a zero angle at all times, this results in an estimate that is quite reliable during fast maneuvers. However, during constant velocity flight the output of the estimator is approximately zero, since it has no absolute measurement of the real angle. This shows that the network uses the accelerometer as an absolute reference to the global attitude, just like the comparison filters do.

2.3.4. Energy consumption and evaluation frequency on neuromorphic hardware

To evaluate the potential of deploying these networks on neuromorphic processors, the update frequency and energy consumption of the network were examined. Since gathering energy benchmarks on the Kapoho Bay, used on our quadrotor, is not possible, the results shown in Figure 2.9 are obtained with a Nahuku board that has 32 Loihi chips by utilizing the energy and execution time probes as provided in the NxSDK. The average execution time of the spiking recurrent layer on hardware was only 10μs, which corresponds to 100 000 Hz. As

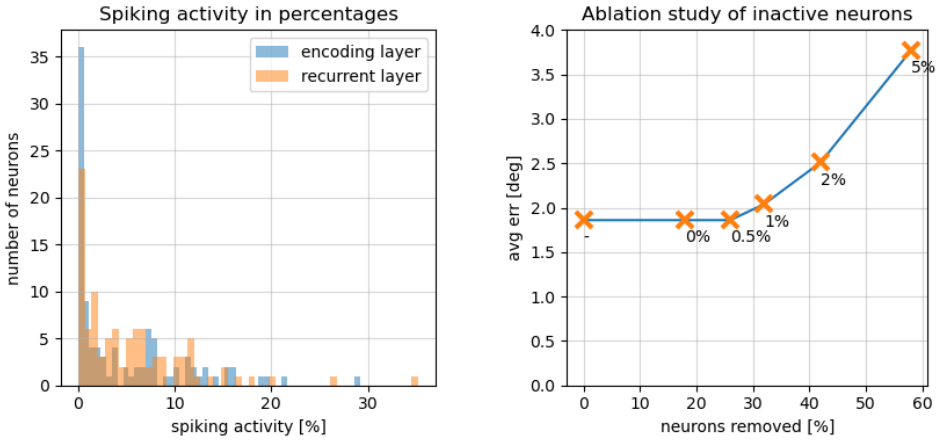


Figure 2.7: Investigation in the spiking activity of all neurons in the Att-SNN and the effect of removing sparsely spiking neurons. A histogram of the spiking activity, averaged over all datasets, is shown on the *left*. The effect of removing sparsely spiking neurons on accuracy to reduce the overall network-size and prospective energy consumption in an end-to-end solution on the *right*.

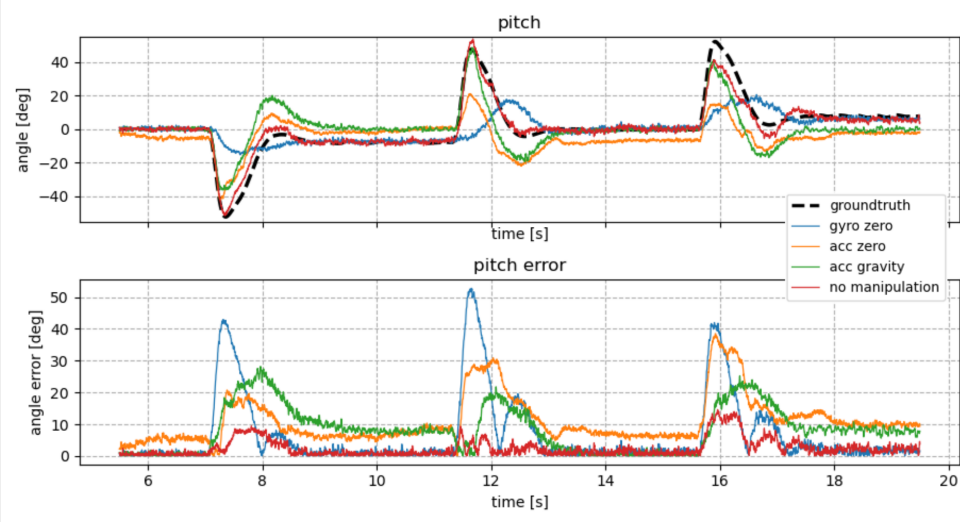


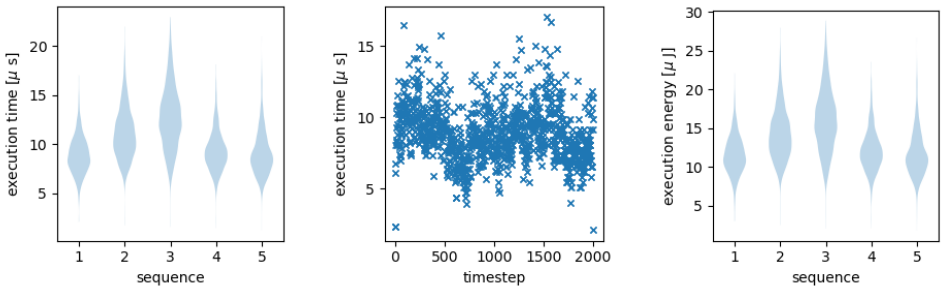
Figure 2.8: Response of the SNN for different types of input manipulation. Both the pitch estimation and error of the network is shown for (i) no data manipulation, (ii) all gyroscope values zero, (iii) all accelerometer values zero, and (iv) all accelerometer values pointing towards gravity in world-frame ($\theta_k^{\text{acc}} = 0$).

comparison, the execution time of the layer was also evaluated using a PyTorch implementation on a laptop with a Nvidia GTX 1650Ti and a 12-core Intel i7-10750H CPU. The average execution time for a single timestep of the Att-SNN here was 555μs, $\approx$55$\times$ running the network on the neuromorphic processor. Also the execution time of the Att-RNN was measured in the same way, and the average execution time here was 150μs, still an order of magnitude higher than on the neuromorphic processor. Also the number of Floating-point Operations (FLOPS) per method per timestep have been calculatd. The Att-SNN requires only $(M^2 + MN + 2M)$ FLOPS, with M the number of neurons in the hidden layer, and N the amount of inputs to this layer. The Att-RNN with gated-recurrent units requires approximately $(3M^2 + 3MN + 3M)$ FLOPS for a single hidden layer, which is almost 3 times more. The average energy consumed per time step was 13.2μJ. In our hardware architecture with the PX4 mini flight-computer connected to an UP Squared computer using Fast-RTPS, we were able to run it at the frequency of incoming IMU data, 200Hz. It should be noted this looprate is dependent on the data transferring in the Von Neumann processors. In an end-to-end neuromorphic pipeline, this could be orders of magnitude faster. Since we run the Loihi at this frequency, the latency from input sample to estimate is governed by the delays in the network. In our implementation, this means an estimate is produced after three timesteps, resulting in a latency of 15ms.

2.4. Discussion

This work demonstrates that a small recurrent spiking neural network is able to estimate the attitude of a quadrotor in flight. In terms of accuracy, it is competitive with filters commonly used in flight-controllers, such as the Mahony-filter. Our approach is not data hungry, as common AI algorithms are, and only requires data obtained in simulation (easily accessible, and available open-source) supplemented with short sequences obtained with a real quadrotor (for our training, less than 20 minutes of real data was used). Our experiments have shown that little data already suffices for successful attitude estimation in nominal flight conditions. However, future work could consider larger datasets with a larger variety of flight maneuvers, and naturally a larger range of sensor biases. This would help to find out how well trained networks generalize to unseen conditions.

These results are obtained with focus on achieving a small-scale network. By increasing the size of the network, the training performance can be increased but



(a) Execution time for 5 sequences of 2000 time steps

(b) Execution times for an example sequence

(c) Energy consumption for 5 sequences of 2000 time steps

Figure 2.9: Energy and evaluation frequency of the Att-SNN on the Nahuku board that has 32 Loihi chips.

the network might suffer more from overfitting. Already in our limited sized network, it was visible that training allowed for overfitting on the training data. This effect might be limited by supplying the training with more samples from different real world quadrotors or making the network more resilient to biases in the input (such as constant offsets of the measured accelerations) by adding parametrization. This parametrization could be obtained by adding adaptivity to the synaptic connections, for which a suitable online learning rule could be designed. Besides, minimizing size is in line with our goals of obtaining miniature-scale robots, capable of performing autonomous missions. Increasing the size too much will also constrain the learning, due to an increased dimensionality, resulting in high memory usage and training time.

A main potential criticism on our work could be that the plain complementary filter, achievable with only a couple lines of code in a microcontroller, is still a valuable choice for performing the task of attitude estimation based on IMU-data. Executed on a widely available and cheap micro-controller, it estimates states fast and with little energy expenditure. However, we aim for a fully neuromorphic pipeline, so that all processing, from very intensive visual processing to less intensive state estimation and control, can happen on a single neuromorphic chip. Hence, we also need to perform attitude estimation with an SNN. Moreover, from a scientific viewpoint, we are interested in understanding how (spiking) neural networks solve this task, potentially unveiling new strategies or delivering new hypotheses on sensor fusion in flying animals like small insects. Already in this study, the SNN showed the interesting property of converging quicker to the attitude when not initialized at the ground-truth attitude. In future work, we plan

to delve deeper into the detailed workings of SNNs estimating attitude.

Currently, the work has the limitation that the Att-SNN only estimates the angles necessary for position control, and excludes yaw. Synthesizing an unbiased, non-diverging yaw-estimate using angular velocities and linear accelerations exclusively is not possible, so extra sensors would need to be included. A common option is the 3-dof magnetometer, measuring the Earth's magnetic field and therefore supplying the estimator with an absolute measurement of the quadrotors heading. Although these sensors are very useful in large drones flying at great altitudes in non-urban environments, the readings are heavily influenced by disturbances in the magnetic field. these disturbances can be caused by electronic appliances in the vicinity or even by the motors of the quadrotor itself. In future work, adding data from other sensors or the output of a controller will be studied to capture a full 3d model of a quadrotor in flight. This will pose interesting new challenges such as dynamics changing over a flight because of a draining battery that might be addressed by an SNN that features adaptivity.

The frequency of the network in this study was chosen to be 200Hz, since this matched the output of the sensor data from the PX4 flight-controller. In operation, the input-data streamed to the network has to be of the same rate as during training, because of the time-constants that characterize the neuron dynamics. During this research, it was found that training on even higher data rates, further reduced the estimation-error. Together with the promises of extremely high update rates of neuromorphic hardware, this is encouraging for further research.

Considering the recurrent layer encodes the attitude (or rate of change of the attitude) necessary for control, the decoding layer can be replaced by another spiking layer that can be trained to perform control tasks, either by supervised learning mimicking a baseline controller, or it can be trained in a RL framework, as is discussed in [9]. These are the next steps that follow naturally from our research.

2.5. Conclusion

In this paper, we have presented an SNN model, called Att-SNN, that can be employed as an attitude estimator in an end-to-end neuromorphic control pipeline. This implementation builds upon three contributions. First, we have shown that the Att-SNN can perform state estimation tasks in highly dynamic systems such as MAVs, with competitive performance when compared to state-of-the-art non-neuromorphic methods and conventional recurrent ANNs. Second, we successfully implemented the Att-SNN on the Loihi neuromorphic

processor, showing outstanding energy and time efficiency, therefore paving the way towards fully embedded neuromorphic control onboard MAVs. Third, this study shows for the first time that these networks can be used as an estimator in the control-loop, showing that small errors do not accumulate over time. Furthermore, our work shows an efficient method of encoding floating point sensor data into binary spikes without having to study the tuning curves of rate-coded neurons or finding the optimal distribution of neurons in population coding. Several prospective future research directions are recognized. This includes extending the estimation with neuromorphic control and reducing the effect of uncertainties due to sensor noise by employing online adaptivity. All together, the Att-SNN neuromorphic attitude estimation model will help closing the neuro-biologically-inspired control loop from sensor to actuator in critical embedded systems such as MAVs.

2.6. Acknowledgements

This material is based upon work supported by the Air Force Office of Scientific Research under award number FA8655-20-1-7044. The authors also wish to express their thanks to J. Hagenaars and F. Paredes-Valles for their fruitful discussions regarding implementing a Loihi-ready quantized version of the network.

2.7. Supplementary materials

All code used in this chapter is available at
`https://github.com/tudelft/neuromorphic_attitude_estimation`.
The data used for training can be found at
`https://doi.org/10.4121/20464830.v1`

Link to paper:

3

Learning Flight Attitude from Vision Alone

Vision is an essential part of attitude control for many flying animals, some of which have no dedicated sense of gravity. Flying robots, on the other hand, typically depend heavily on accelerometers and gyroscopes for attitude stabilization. In this chapter, we present the first vision-only approach to flight control for use in generic environments. We show that a quadrotor drone equipped with a downward-facing event camera can estimate its attitude and rotation rate from just the event stream, enabling flight control without inertial sensors. Our approach uses a small recurrent convolutional neural network trained through supervised learning. Real-world flight tests demonstrate that our combination of event camera and low-latency neural network is capable of replacing the inertial measurement unit in a traditional flight control loop. Furthermore, we investigate the network's generalization across different environments, and the impact of memory and different fields of view. While networks with memory and access to horizon-like visual cues achieve best performance, variants with a narrower field of view achieve better relative generalization. Our work showcases vision-only flight control as a promising candidate for enabling autonomous, insect-scale flying robots.

Parts of this chapter were under review at time of writing this thesis [42]

3.1. Introduction

Attitude control is a fundamental challenge in aerial robotics. For drones to execute their missions, they must precisely control their orientation relative to gravity—a task traditionally addressed by IMUs (inertial measurement units) that provide absolute acceleration and rotation rate measurements [43]. Yet, flying insects exhibit remarkable flight agility without any known sensor dedicated to measuring gravity [44]. Flying insects with four wings such as honeybees even lack the halteres that provide two-winged flying insects with direct sensory information on rotation rates [45]. Previous work has hypothesized that flying insects can in principle rely on visual cues alone to estimate flight attitude [46]. Specifically, it was shown that optical flow can be combined with a motion model to estimate and control flight attitude. Besides insect understanding, this insight opens a pathway toward lighter and potentially more robust flying robots. By eliminating the dependency on IMUs, ultra-lightweight sensor suites [47] could be made even lighter, simpler, and more efficient. This work demonstrates a general approach to flight control without IMU—bringing tiny autopilots and insect-scale flying robots one step closer.

Vision-based attitude estimation for flying robots goes back to early work on horizon-line detection methods, applied to fixed-wing drones flying high in wide open environments [48, 49]. Later, methods have been developed that rely on the specific structure of human-made environments. Assuming parallel lines in view, vanishing points can be determined and used as attitude estimators [50, 51]. However, flying insects are also able to control their attitude in unstructured environments where the sky is not visible, a property that is also of interest for flying robots. Combining optical flow with a motion model enables attitude estimation in such generic environments, only relying on sufficient visual texture. De Croon et al. [46] show that attitude can be inferred from optical flow when combined with a motion model that relates attitude to the direction of acceleration. However, due to hardware-related update-rate limitations, their real-world flight demonstrations still rely on gyroscope measurements.

Event-based cameras offer a promising solution to these limitations [20] with their low latency, high temporal resolution, and robustness to motion blur. Existing work on estimating rotation rates with event cameras [52–54] has so far been limited to motions with little translation-rotation ambiguity (rotation-only or restricted motion like driving). Furthermore, work on estimating flight attitude from vision has been limited in environmental complexity (requiring a structured environment with vanishing lines) [55] or still requires the use of gyroscopes for

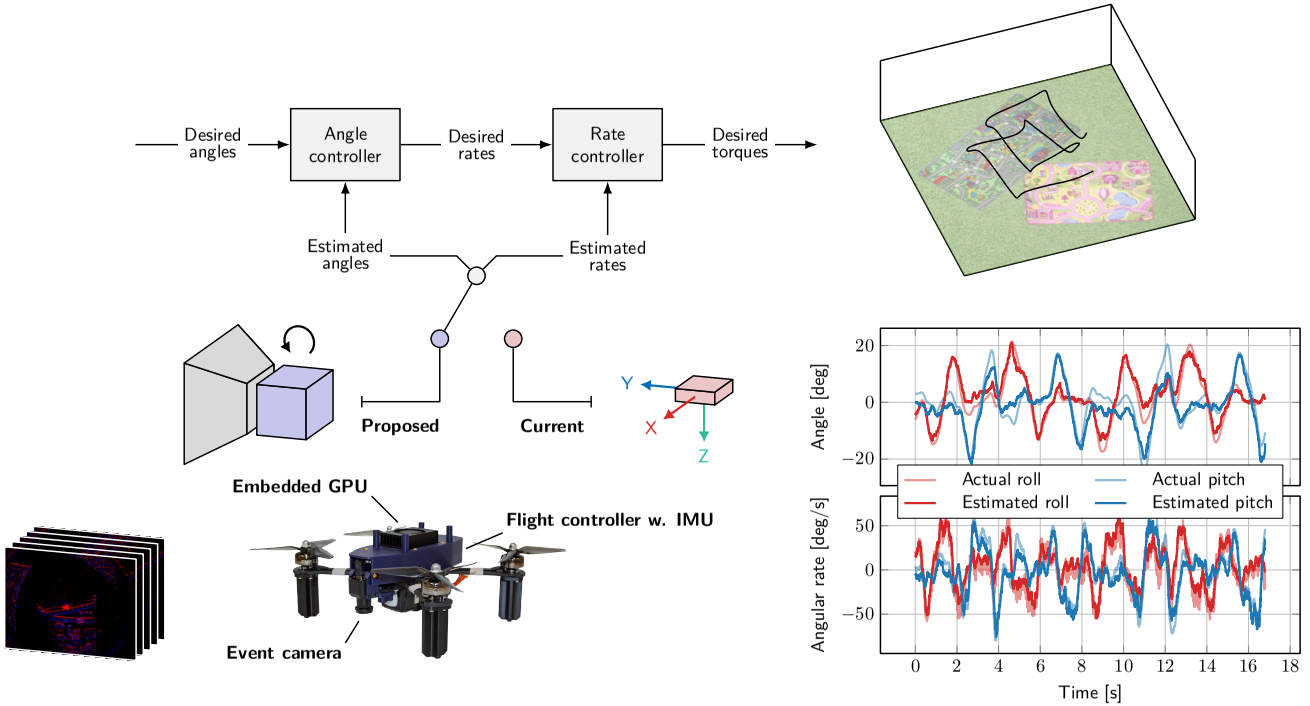


Figure 3.1: Vision-only flight control promises lighter flying systems. We demonstrate that fully on-board, vision-only flight control is possible with a low-latency vision pipeline consisting of a small recurrent convolutional neural network (ConvRNN) and an event-based camera. Left: Our proposed pipeline takes the place of the IMU (inertial measurement unit) in a traditional flight control loop, estimating both attitude and rotation rates. Right: The resulting system demonstrates accurate estimation and control in real-world flight tests. The plots show both the actual attitude angles and rates (as measured by the flight controller) and the estimated ones (predicted by a neural network).

low-level flight control [56–58]. Thus, a critical gap remains: achieving fully on-board, IMU-free attitude control in realistic flight conditions.

In this paper, we address this gap by demonstrating—for the first time—a fully on-board flight control system capable of estimating both attitude and rotation rate solely from event-based visual inputs. Specifically, we develop a recurrent convolutional neural network trained through supervised learning to map raw event streams directly to accurate attitude and rotation rate estimates. Unlike traditional approaches, which use IMU measurements in combination with a filter to explicitly model the drone's motion, we take a learning approach which trains a neural network to implicitly acquire the relation between visual cues and the vehicle's state. Opting for learning a neural network means our estimator can eventually be integrated in an autopilot which is learned end-to-end as a neural network. This will be especially relevant if not only event-based vision but also neuromorphic computing is used on the flying robot [59, 60].

We demonstrate, with real-world flight, that our combination of event camera and low-latency neural network can replace the traditional IMU and filter combination in the flight control loop. Furthermore, we investigate the learned network's generalization to different environments, and compare alternative network inputs and architectures. We show that neural network memory and a wide field of view are essential components for accurate estimation, but that greatly reducing the field of view, and denying the network of most horizon-like visual cues, leads to improved relative generalization across environments. While this hints at the learning of an internal model for attitude from visual motion, it comes at the cost of reduced absolute performance.

The contributions of this work can be summarized as follows:

1. The first fully on-board, vision-only and IMU-free pipeline for control of a real-world, unstable quadrotor.

2. A low-latency recurrent convolutional neural network capable of estimating flight attitude and rotation rate from event camera data through supervised learning.

3. An investigation into the approach's performance, necessary components, and generalization to different environments.

Our work promises extremely efficient, end-to-end-learned autopilots with minimal sensors, capable of powering the next generation of insect-scale flying robots.

3.2. Results

3.2.1. In-the-loop flight tests

3.1 gives an overview of our proposed system. We integrate an event camera and small recurrent convolutional neural network running on an on-board GPU into the drone's flight control loop. The network estimates attitude and rotation rates from event camera data with low latency, acting as a stand-in replacement of a regular IMU (inertial measurement unit), and allowing for accurate control. In traditional flight control loops, IMUs measure linear accelerations and rotation rates at high frequency (typically $\gg$ 500 Hz). These measurements are subsequently integrated in a Kalman or complementary filter running at a lower frequency (100-200 Hz) to produce accurate attitude and rotation rate estimates while filtering out the high-frequency noise produced by the sensors.

The proposed system instead uses data from an event camera in combination with a learned estimator. Events are accumulated into frames of 5 ms, and fed to a recurrent convolutional neural network that then estimates attitude and rotation rate. These estimates are sent to the flight controller at 200 Hz, and used by the regular angle and rate controller to control the drone. We train our network through supervised learning on a dataset containing events, along with attitude and rotation rates from the flight controller as labels. While these are not absolute ground truth (they are estimated by the flight controller using the IMU), they are typically reliable but can have bias (which is dealt with by subsequent integrators).

The results from several flight tests with the trained network in the loop are shown in Fig. 3.2. For these tests, the pilot commands the drone to hold its position, which the flight controller translates to attitude setpoints with the help of a higher-level position controller and a velocity sensor. These attitude setpoints are subsequently compared against estimates provided by the neural network. This results in desired rotation rates, which are also compared against network estimates, resulting in desired torques sent to the motors. Fig. 3.2 contains flight tests in which the drone is commanded to stay in the same place.

The traces and histograms show that the network can accurately estimate both attitude and rotation rate, with most errors within ± 3 deg and ± 18 deg/s for approx. 10 minutes of flight. The error plots for different attitude/rate combinations show that most errors are due to underestimation by the network, which can also be seen in the difference between network and ground truth during the transition phase. Underestimation is most pronounced at high angles/rotation rates, which could be explained by the inertia of the network's memory. While this allows integration of

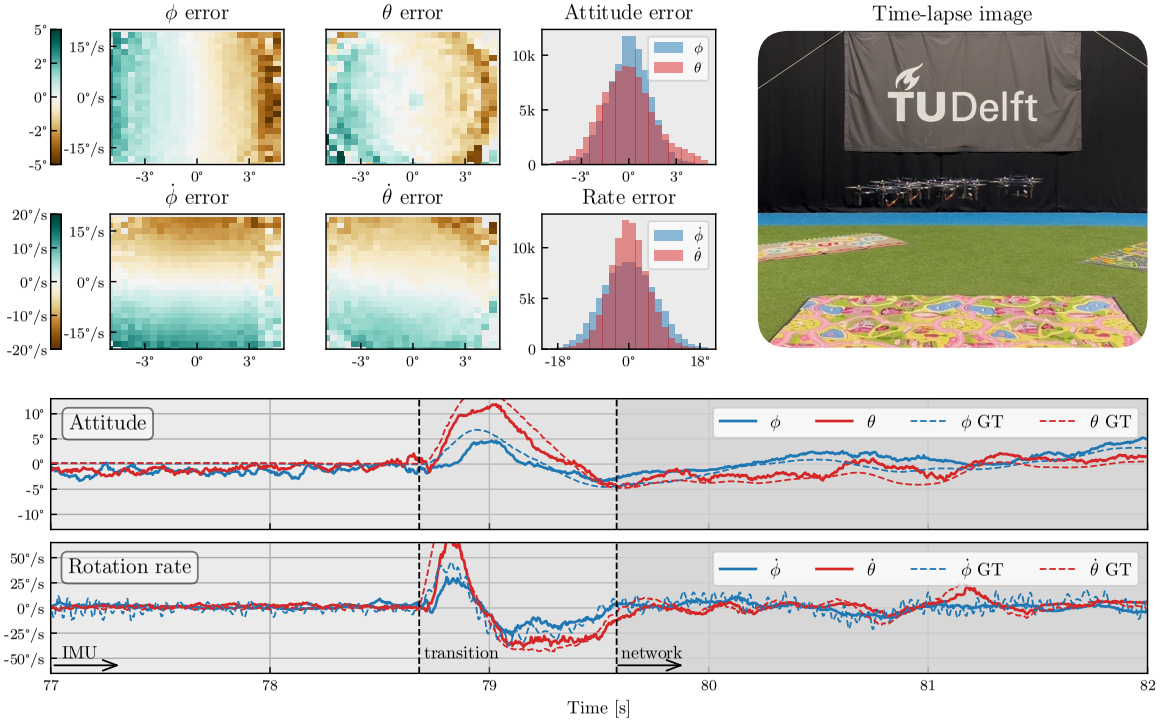


Figure 3.2: In-the-loop hover (position hold) tests with the network in control. The right-most time-lapse image shows the variation of the drone during flight, demonstrating the controllability of the drone over a total of approx. 10 minutes of flight time. The other plots quantify the errors of the estimated attitude and rotation rate for roll (rate) ϕ ($\dot{\phi}$) and pitch (rate) θ ($\dot{\theta}$). We consider the flight controller estimator as ground truth (GT). The histograms give the error distribution of the network estimate compared to ground truth, with the biases of both subtracted (to disregard biases in training data). The left-most plots show the network errors for different attitude/rate combinations. These show that the network for both attitude and rate underestimates the real value. The bottom traces show the transition from IMU to network control, with a reset of the flight controller's integrator causing a short response of the drone before settling.

information over time, it also limits the network in following fast maneuvers.

The error histograms further reveal axis-dependent asymmetries: attitude estimates are more accurate for roll than for pitch, whereas the opposite holds for rotation rate estimates. We attribute reduced accuracy in pitch attitude estimation primarily to limited pitch-angle variability during training and minor shifts in drone balance along the pitch axis (such as varying battery position). Conversely, the decreased accuracy observed in roll rate predictions aligns with the higher noise levels present in the ground-truth roll rate measurements coming from the flight controller.

3.2.2. Comparison of models

We conducted an extensive comparison of neural network models with varying network architecture, input modalities and input resolution. Table 3.1 lists the performance of these variations in terms of RMSE (root mean square error) and MASD (mean absolute successive difference) when tested on unseen data from the training environment. While RMSE gives a good measure of the estimation error, MASD quantifies prediction smoothness by looking at the difference between subsequent predictions. A qualitative illustration of the estimation performance of various neural networks is given in Fig. 3.3.

The baseline, *Vision*, receives only event frames as input and processes these with a convolutional neural network with GRU (gated recurrent unit) memory block. This variant was used for the in-control flight tests in Fig. 3.2. *VisionMotor* additionally takes motor speeds as input. These can serve as a proxy for the moment generated by the motors, (a prediction of) which was shown to be necessary for the attitude to be observable [46]. In our learning setup, however, the error difference with the vision-only model is small: only the estimation of rotation rates is slightly better, which makes sense given that the forces produced by the motors can be integrated to obtain rotational velocities. *VisionGyro* receives gyro measurements (rotation rates) as additional inputs. This represents the case in which a gyro would still be present in the system, and its performance can give an idea of whether the network could integrate rotation rate to obtain the attitude. As expected, this results in the lowest-error rotation rate estimates. The accuracy of the attitude prediction, however, is not meaningfully better. *VisionFF* replaces the recurrent memory block with a feedforward alternative, and will therefore not be able to integrate information over time. While attitude can be inferred from a single image by looking at horizon-like visual cues (such as those indicated by the white arrows in Fig. 3.5), the increased attitude error of

VisionFF compared to *Vision* suggests that having memory is still beneficial. Estimation of velocities such as rotation rate is very difficult without memory, and this is reflected by the large increase in rotation rate error.

VisionSNN is a hybrid spiking neural network (SNN) where the encoder has binary activations (stateless spiking neurons) and the recurrent memory block consists of spiking LIF (leaky integrate-and-fire) neurons with a recurrent connection. While the quantitative errors for *VisionFF* and *VisionSNN* are similar, Fig. 3.3 shows that the SNN's memory makes a difference for estimating rotation rates.

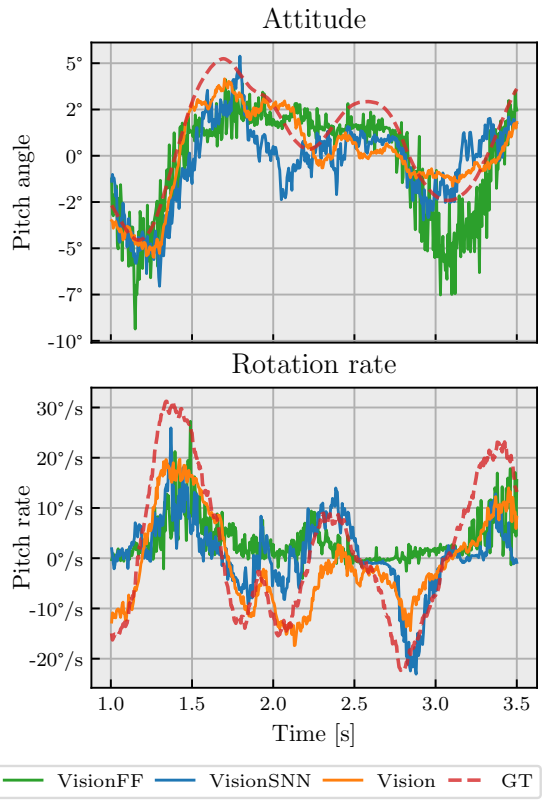


Figure 3.3: Qualitative comparison of attitude and rotation rate estimates for various neural network architectures on unseen data from the training environment. The network variants match those in 3.1.

Next, we investigate the impact of varying input resolution. Under rapid motion, event cameras generate a large amount of events, which can lead to bandwidth saturation and increased latency when working with on-board, constrained hardware. The event camera used in this work, a DVXplorer Micro, allows

	Attitude		Rotation rate	
Network	**RMSE [deg]**	**MASD [deg]**	**RMSE [deg/s]**	**MASD [deg/s]**
Vision	1.51	**0.27**	10.65	2.57
VisionMotor	1.64	0.31	9.57	2.47
VisionGyro	**1.47**	0.31	**4.01**	**2.36**
VisionFF	2.17	1.00	18.65	5.82
VisionSNN	2.17	0.52	15.03	4.04

Table 3.1: Performance of models with different network architectures and inputs on unseen data from the training environment. *Vision* is the baseline model with ConvGRU memory and vision-only input. *VisionMotor* additionally has motor commands as input. *VisionGyro* receives gyro measurements as additional input. *VisionFF* has a memory-less architecture (feedforward). *VisionSNN* is a hybrid spiking neural network. We quantify performance in terms of RMSE (root mean square error) and MASD (mean absolute successive difference).

disabling pixels to limit the number of events generated by the camera. Fig. 3.4 analyzes the impact of using lower-resolution data on network performance when training with otherwise identical settings. Apart from quantifying the estimation error using RMSE, we also look at the prediction delay of the network. When actively controlling a system, significant delays lead to oscillations and potentially instability, and should therefore be avoided. We quantify the prediction delay of each network by looking at the time shift for which the Pearson correlation coefficient (PCC) is lowest. The results show that there is a noticeable delay in predictions when using only a quarter of the camera's pixels. We attribute this to the fact that attitude changes might only be seen by any of the enabled pixels once they become large enough, leading to a delayed response. This delay is much less present for half and full-resolution networks. Interestingly, the full-resolution network shows a slightly higher attitude RMSE compared to the half-resolution network. This could be explained by the fact that all networks were trained for the same number of epochs, whereas the larger full-resolution network likely requires more training steps before achieving similar convergence. The half-resolution network brings a good balance between computational efficiency and estimation performance.

3.2.3. Generalization and internal motion model

Recent work [46] has shown that attitude can be inferred from optical flow when combined with a motion model relating attitude to acceleration direction. While

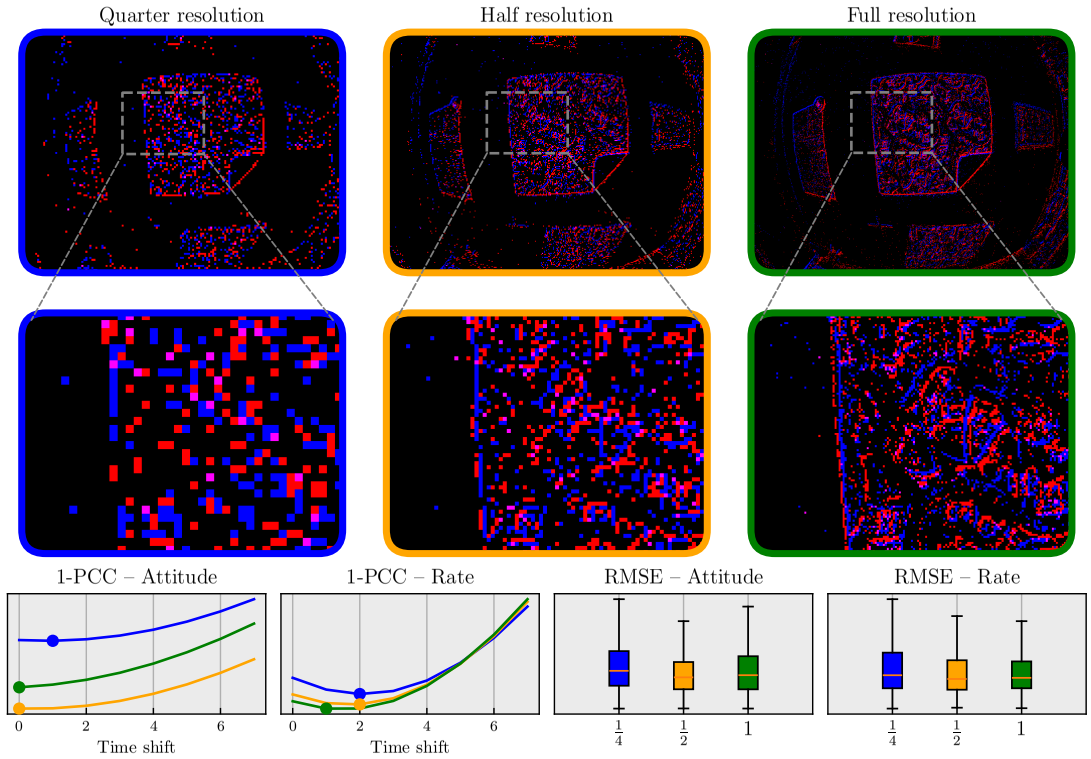


Figure 3.4: The impact of resolution on network performance after training under otherwise identical settings. Results were obtained by enabling only a subset of event camera pixels: every fourth pixel (quarter resolution, blue), every other pixel (half resolution, orange) and all pixels (full resolution, green). The bottom-left graphs show the PCC (Pearson correlation coefficient) for different time shifts of the prediction targets. The minimum of each line indicates the shift with the highest correlation. Minima at larger shifts indicate a delay in the network prediction. The bottom-right plots show the RMSE (root mean square error) on validation data. Networks trained on quarter-resolution data show larger prediction delay and increased error compared to higher resolutions.

the learning framework presented here does not explicitly represent such a model internally, it would be interesting to investigate whether this can be promoted during learning, and whether this affects generalization to different scenes. In other words, we would like to steer the network towards using optical flow for attitude estimation, instead of static visual cues such as horizon-like straight lines on the edges of the field of view (indicated by white arrows on the right in Fig. 3.5). We hypothesize that networks that rely mainly on motion features generalize better across environments than networks that focus on visual appearance, which is more scene-specific and may lead to overfitting on the training scene.

To achieve this, we trained a network on the same dataset as before (CyberZoo), but restricted its input to a small 160×120 center crop. This eliminates most visual cues that contain absolute attitude information while preserving motion cues. We refer to the original, unrestricted model as *full FoV* (identical to *Vision* from Table 3.1) and the newly trained variant as *center crop*. We evaluate these models on four unseen sequences, and show the results in Fig. 3.5. The full-FoV network effectively makes use of the horizon-like cues present in most sequences (white arrows for CyberZoo, Office and Outdoor), generalizing well to other scenes and outperforming the center-crop model. However, we also include a sequence from the event camera dataset ECD `poster_rotation` recording [61]. Here, a camera (different from ours) looks at a planar poster while undergoing rotations, without any visual cues related to the camera's attitude. On this sequence, the center-crop network achieved lower attitude and rate errors than the full-FoV network. This indicates that the center-crop network can effectively use motion information to infer attitude, hinting at an internally acquired motion model. Furthermore, looking at the relative error across environments, we see that relative error increases less for the center-crop network when moving from the familiar training environment to novel scenes. This presents a trade-off: while networks with access to the full FoV have better absolute performance for scenes with horizon-like visual cues, networks forced to infer attitude from motion through a reduced FoV relatively generalize better across environments. If we remove memory from the network (*center crop* + *feedforward*), it performs poorly for both attitude and rotation rate estimation. Such a network has neither the field of view for static attitude information, nor the ability to build it up through memory.

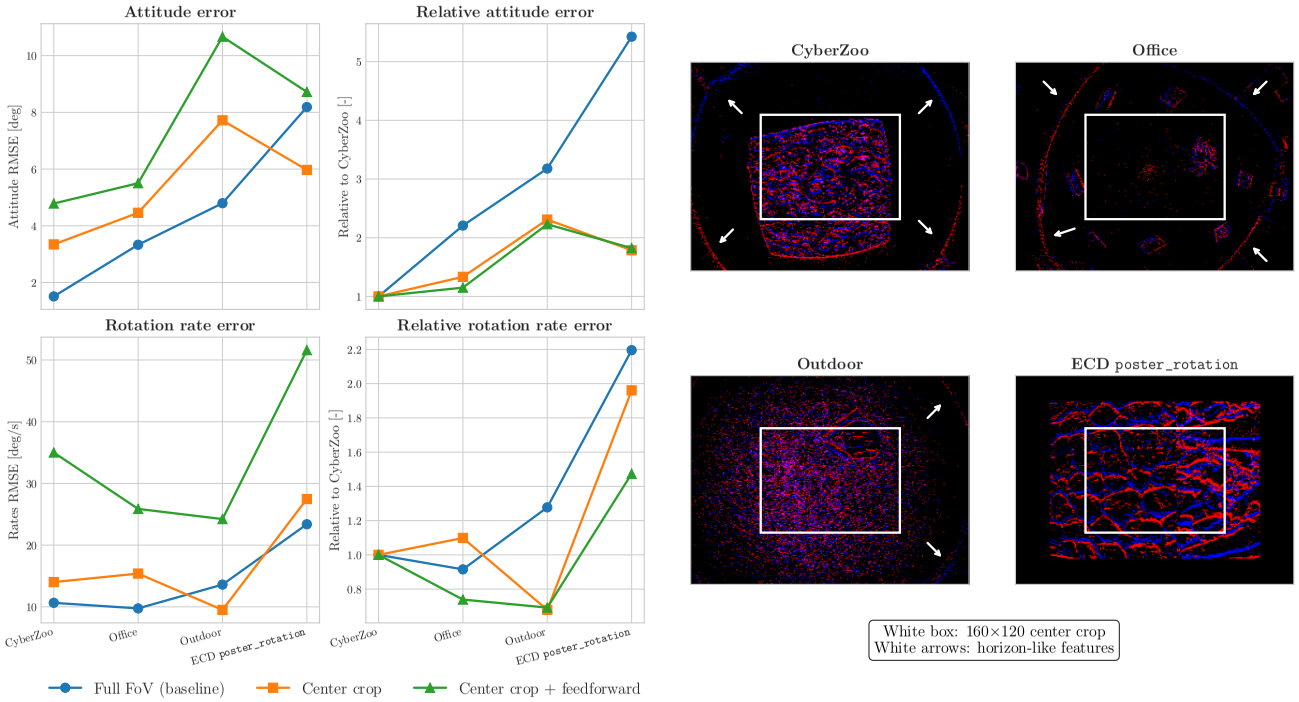


Figure 3.5: Comparison between a network with memory and full field-of-view (baseline), a network with memory that only sees a center-cropped portion (indicated by the white rectangle), and a network without memory that only sees a center-cropped portion. We compare estimation errors for four unseen sequences. CyberZoo is the same environment as trained on, but Office, Outdoor and ECD `poster_rotation` (rotation only) [61] are unseen environments. While the full-FoV network performs better in scenes where horizon information (indicated by white arrows) is available to provide an absolute indication of attitude (CyberZoo, Office, Outdoor), the center-cropped network performs better when there are no visible horizon-like cues (as in ECD `poster_rotation`), hinting at the use of an internal motion model. Additionally, the smaller relative increase in error in the case of the center-cropped network for scenes different from the training location CyberZoo indicates improved generalization. A memory-less center-crop network is unable to aggregate information internally into any kind of model, and hence performs poorly in terms of both attitude and rotation rate estimation.

3.3. Discussion and conclusion

In this work, we presented the first demonstration of stable low-level flight control based purely on visual input, eliminating the need for an IMU (inertial measurement unit). By combining an event camera with a compact recurrent convolutional neural network, we achieved real-time attitude and rotation rate estimation, enabling closed-loop control without inertial sensing. Leveraging the event camera's low latency and high temporal resolution, our vision pipeline delivers the responsiveness required for agile flight, running entirely on-board at 200 Hz.

Removing the IMU simplifies hardware, reducing weight and complexity—both critical considerations for small, bio-inspired flying robots—and our work shows that it is possible to estimate and control flight attitude based on vision alone. Such a vision-only control pipeline offers key advantages for aerial robotics: insect-scale flying robots will already rely on visual inputs for navigation, so with our proposed method no extra sensors would be necessary. Processing could run on a tiny energy-efficient, possibly neuromorphic, processor. Our experiments show that estimation performance generalizes to unseen and visually distinct environments, and that this can potentially be improved further by forcing the network to infer attitude from visual motion instead of horizon-like appearance cues. Comparisons between network architectures further highlight the importance of memory: while feedforward networks can infer static attitude from scene appearance, recurrent models are crucial for accurately tracking dynamic flight states, underlining the role of memory in enabling robust vision-based control.

Several limitations remain. We observed systematic underestimation of both attitude and rotation rates, with axis-dependent asymmetries. These can be attributed to biases in training data distribution, particularly in pitch motions, and to shifts in drone balance. Reducing these biases—through more diverse datasets, improved calibration or higher-accuracy ground-truth (such as from a motion capture system)—could improve overall performance. Additionally, we found that lowering input resolution introduces prediction delays, while higher resolutions impose greater computational demands without proportional gains in performance. A half-resolution setting emerged as a practical trade-off for real-time operation.

Future work should focus on increasing the robustness of learning and further hardware integration. Making use of the contrast maximization framework for self-supervised learning from events [62] would allow direct learning of a robust

and low-latency estimator of optical flow decomposed as rotation and translation [59], which could be integrated with a learned visual attitude estimator to give the most robust estimate. Hardware could further be integrated through a combined event camera and neuromorphic processor setup, such as the SynSense Speck [63], to achieve lower power consumption, weight and latency. Furthermore, a deeper investigation into the internal motion model developed by networks may yield deeper insights into the mechanisms they use to solve the task at hand. We have shown that parts of the network generalize across scenes, but also that the networks can exploit scene-specific features that resemble horizon-like lines to get an estimate of the attitude. Such insights could enable the design of more robust and autonomous robotic systems that rely heavily on environmental perception.

Overall, our results demonstrate that vision-only attitude estimation and control is a viable alternative to traditional inertial sensing. By simplifying the sensor suite and removing the dependency on IMUs, our approach paves the way for the next generation of lightweight, agile, and bio-inspired flying robots.

3.4. Methods

3.4.1. Estimating attitude and rotation rate from events

The analysis by De Croon et al. [46] shows that a drone's flight attitude can be extracted from optical flow when combined with a motion model. Many conditions are analyzed. In the simplest case, they derive local observability from the ventral optical flow component ω_y for roll ϕ and a simple motion model that relates attitude angles to acceleration direction:

$$\omega_y = -\frac{\cos^2(\phi) v_y}{z} + p \tag{3.1}$$

$$f(\boldsymbol{x}, u) = \begin{bmatrix} \dot{v}_y \\ \dot{\phi} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} g \tan(\phi) \\ p \\ 0 \end{bmatrix} \tag{3.2}$$

where state $\boldsymbol{x} = [v_y, \phi, z]^T$, control input $u = p$, z represents the height above ground, v_y is the velocity in the body frame's y-direction, and p denotes the roll rate. This relationship holds for $\phi \in (-90°, 90°)$, and the same derivation can be made for ω_x and pitch θ. Although this model assumes constant height, they show extensions for changing height and uneven/sloped environments. In those, the divergence of the optical flow field can be used to observe variation in height, maintaining the partial observability of the system. The system is only partially

observable, since attitude becomes unobservable at angles and rates close to zero, for instance when the drone is hovering in place. Nevertheless, the parts of the state space that make the system unobservable are inherently unstable and drive the system to observable states, thus closing the loop.

While Eq. 3.1 and Eq. 3.2 treat the rotation rate p as a known control input (from gyro measurements), this is theoretically not necessary, and a prediction of the moments M resulting from control inputs suffices [46]:

$$f(\boldsymbol{x}, u) = \begin{bmatrix} \dot{v}_y \\ \dot{\phi} \\ \dot{p} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} g\tan(\phi) \\ p \\ M/I \\ 0 \end{bmatrix} \tag{3.3}$$

where I is the moment of inertia around the relevant axis. In our approach, motor speeds as input could replace M and the network could learn an internal motion model to obtain attitude. However, this is not guaranteed, and as the successful flight tests with a vision-only network show, not necessary either when using a machine learning approach. Learned models exploit other factors, as may insects. We explore one of these factors (a large field of view) in this article. A large field of view enables proper separation of translational and rotational flow, which allows more accurate extraction of control inputs from vision. As described above, these control inputs allow for the system to be observable.

Training and network details

We train a small recurrent convolutional network to estimate flight attitude and rotation rate from events. Training is done in a supervised manner, with ground-truth attitude and rate coming from the flight controller's state estimation. We collect training data containing diverse motions and attitudes in our indoor flight arena. We make use of two-channel event count frames as input to the network, with each frame containing 5 ms of events. For training, we take random slices of 100 frames from a sequence, and use truncated backpropagation through time on windows of 10 frames without resetting the network's memory. This is done to get a network that (i) accumulates temporal information internally for proper rate estimation, and (ii) can keep estimating stably beyond the length of the window it was trained on. We train until convergence using an MSE (mean squared error) loss, Adam optimizer and a learning rate of 1e-4. We add a weight of 10 to the attitude loss to make it similar in magnitude to the rate loss. Data augmentation consists of taking random slices of frames, randomly flipping event frames (and labels accordingly) in the channel dimension (polarity flips), height

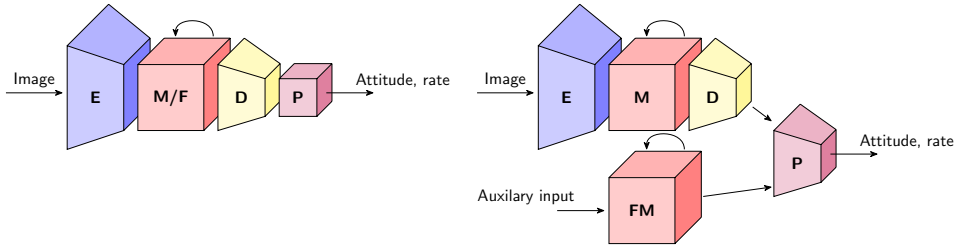


Figure 3.6: Schematic overview of network architectures. Left: baseline vision-only network. Right: network with vision and an auxiliary input (motor speeds, rotation rates). Encoder (E), memory (M) and decoder (D) are convolutional. Flattening to attitude and rotation rate estimates happens in the predictor (P). We swap the memory for a feedforward (F) block to get a network without recurrency. For the auxiliary input, we use fully connected (FM) instead of convolutional memory.

(up-down flips) and width (left-right flips). For evaluation, we run networks on full sequences without resetting, and we evaluate in terms of RMSE (root mean square error) and MASD (mean absolute successive difference):

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.4}$$

$$\text{MASD} = \frac{1}{n-1}\sum_{i=1}^{n-1}|x_{i+1} - x_i| \tag{3.5}$$

The baseline network consists of an 8×-downsampling encoder followed by a GRU (gated recurrent unit) memory block and a decoder transforming the memory into angle and rate predictions. The encoder and memory are convolutional to stimulate the learning of local motion-related features that generalize well, and to prevent overfitting to scene-specific appearance (which we observed when using a fully connected GRU). We experiment with different network variants in terms of receiving extra inputs (drone motor speeds, rotation rates) or having a feedforward instead of recurrent block (no memory). Schematic illustrations of these are shown in fig. 3.6. Apart from the GRU blocks and final output, we use ELU (exponential linear unit) [64] activations throughout the network.

3.4.2. Using estimates for real-world robot control

We use the two-layered control architecture illustrated in fig. 3.1. While it closely follows the implementation in our flight controller, it is running on a separate on-board computer. This allows us to use the control gains from the flight controller

as-is, with only minor tweaking to account for the delay added by communication between the on-board computer and the flight controller.

The first layer consists of a proportional controller that compares commanded attitude and estimated attitude to generate rotation rate setpoints. The second layer is implemented as a PID (proportional-integral-derivative) controller, translating these rotation rate setpoints—combined with estimated rotation rates—into torque commands. These are then sent to the motor mixer running on the flight controller. The motor mixer converts these torque commands, along with a single thrust command, into individual motor commands. The mixing process involves a linear transformation based on the specific geometric layout of the drone.

For our flight tests, we have a human pilot controlling the drone. While stable flight is possible with the pilot immediately commanding the attitude of the drone (angle or stabilized mode), we make use of an outer-loop position controller running on the flight controller (position mode). An additional optical flow sensor provides velocity estimates, allowing the pilot to control position instead of attitude. This greatly simplifies flight testing, and makes individual tests more repeatable. Furthermore, because our training data inherently contains biases, some kind of outer-loop controller (whether a human pilot or a software position controller) is necessary to mitigate these biases during testing.

Robot setup

We use a custom 5-inch quadrotor (shown in fig. 3.1) to perform real-world flight tests. The drone has a total weight of approximately 800 g, including sensors, actuators, on-board compute and battery. All algorithms are implemented to run entirely on board, using an NVIDIA Jetson Orin NX embedded GPU to receive data from the event camera, estimate flight attitude and rotation rate, and calculate control commands in real time. Body torque commands based on the estimated attitude and rotation rate are sent to the flight controller, which is a Kakute H7 mini running the open-source autopilot software PX4. Communication between the flight controller and the embedded GPU is done using ROS2 [65]. An MTF-01 optical flow sensor and rangefinder enables pilot control in position mode. We record ground-truth flight trajectories (for plotting) using a motion capture system.

We use a downward-looking DVXplorer Micro event camera in combination with a 140°-field-of-view lens to capture as much of the environment as possible. To prevent bandwidth saturation while keeping good estimation performance, we only enable every other pixel on the sensor, resulting in a 320×240 stream

(instead of 640×480) for the same field-of-view. These events are accumulated into 5 ms frames for the network. The entire events-to-attitude pipeline is running at approximately 200 Hz, with the embedded GPU consuming around 9 W on average. While the pipeline can run at frequencies as high as 1000 Hz, communication limited real-world flight tests to 500 Hz without any logging, and 200 Hz with logging.

3.5. Data and code availability

Datasets, code and hardware setup instructions are publicly available (upon publication) via the project's webpage:
`https://mavlab.tudelft.nl/attitude_from_events`.

3.6. Acknowledgements

We would like to thank the participants of the 2024 Capo Caccia Neuromorphic Workshop for their discussions and insights. This work was supported by funding from the Air Force Office of Scientific Research (award no. FA8655-20-1-7044) and the Dutch Research Council (NWO, NWA.1292.19.298).

3.7. Author contributions

All authors contributed to the conception of the project and to the analysis and interpretation of the results. J.H. and S.S. built the drone and integrated hardware and software into a stably flying platform. J.H. developed the software for training networks and running them on board the drone. S.S. developed the software for converting network estimates to low-level control commands on the drone. J.H. and S.S. collected datasets for training and performed flight tests to gather results. J.H. and S.S. wrote the first draft of the article. All authors contributed to the reviewing of draft versions and gave final approval for publication.

Link to paper:

4

Control through fixed network connectivity

Using spiking neural networks, neuromorphic hardware can be leveraged for outstanding update rates and high energy efficiency for robotic control tasks. Yet, low-level controllers are often neglected and remain outside of the neuromorphic loop. Designing low-level neuromorphic controllers is crucial to remove these traditional controllers, and therefore benefit from all the advantages of closing the neuromorphic loop.

In this chapter, we propose a parsimonious and adjustable neuromorphic PID controller, endowed with a minimal number of 93 neurons sparsely connected to achieve autonomous, onboard altitude control of a quadrotor equipped with Intel's Loihi neuromorphic chip. We successfully demonstrate the robustness of our proposed network in a set of experiments where the quadrotor is commanded to reach a target altitude from take-off. Our results confirm the suitability of such low-level neuromorphic controllers, ultimately with a very high update frequency.

Parts of this chapter have been published in The Proceedings of the International Conference on Neuromorphic Systems (ICONS) 2022 [66]

4.1. Introduction

In the coming years, autonomous drones are expected to perform a wide range of complex tasks in unknown environments. These tasks include autonomous take-off and landing, dynamic obstacle detection and avoidance, long-range navigation, etc [67]. In spite of all the major accomplishments over the last decades in aerial robotics research, drones still cannot compete with their biological counterparts such as flying insects and birds. Indeed, these animals perform similar tasks using less computational power at a higher energy efficiency while being more robust to disturbances like wind gusts. For instance, despite their mere 100,000 neurons, fruit flies nimbly perform aggressive flight maneuvers and chase for mates while avoiding obstacles in complex, cluttered environments [68]. Similarly, desert ants *Cataglyphis* are indisputably champions at long-range navigation in the desert, yet they only have 250,000 neurons to ensure such impressive performance [69]. In contrast, state-of-the-art quadrotors are often equipped with heavy, energy-consuming computing units like Graphics Processing Units (GPU) to enable Artificial Intelligence (AI) based solutions for in-flight autonomy such as vision-based obstacle avoidance in racing tasks [70, 71]. As a matter of fact, the application of conventional neural networks for Micro Air Vehicles (MAVs) is limited by the energy consumption, weight and synchronous nature of the available hardware. This is particularly true in the context of vision-based control where the use of deep Convolutional Neural Networks (CNNs) severely hampers MAVs' flight duration. In addition, the forecast of the end of Moore's law in the coming decades suggests that the pressure on embedded processing will increase with drones' autonomy [72, 73].

This analysis has driven researchers to put efforts in developing novel forms of information representation and processing, such as asynchronous Spiking Neural Networks (SNNs) to more closely model natural neurons and synapses and benefit from their overall performance [74, 75]. These networks offer ample opportunity for a higher energy efficiency and faster computation but require dedicated neuromorphic hardware that can deal with the analog nature of the neuron membrane dynamics. Over the last decade, efforts have been made to develop the first neuromorphic processors to run SNNs, such as HICANN [14], NeuroGrid [15], IBM's TrueNorth [16], APT's SpiNNaker [17] and Intel's Loihi [19]. This new generation of processors yields great opportunities for the application of SNNs in aerial robotics, especially for MAVs where energy, payload and processing time are crucial. Autonomous control for quadrotors requires low latency and high computational power at low energy cost. Such performance can

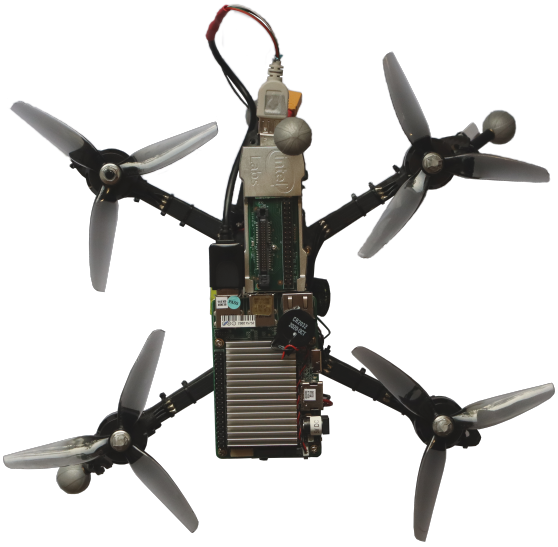


Figure 4.1: Quadrotor used in this research equipped with a neuromorphic processor to perform altitude control.

be achieved by means of neuromorphic algorithms like SNNs running on neuromorphic hardware.

In previous work, we introduced an evolved SNN to control the landing of a MAV equipped with the Loihi chip [23]. Made of only 35 spiking neurons, this model used the divergence of the ventral optic flow to determine the thrust set-point to send to a low-level PID controller running on the von Neuman CPU. In order to make the whole processing neuromorphic and therefore increase the control loop frequency, neuromorphic low-level controllers are required. Examples of such systems have been proposed and include controllers for open- and closed-loop controllers for DC motors [76, 77], robot manipulators [78], optic-flow based landing and neuromorphic implementations of standard PID controllers [26, 79]. The computational and logical simplicity of PID controllers makes them an interesting choice for many control tasks.

In Stagsted et al. [26], the mathematical operations of the conventional PID were implemented in a position-coded SNN by utilizing *operation* arrays. Using such a position-coding, the precision of the controller corresponds to the number of neurons used for the encoding. To obtain an error precision of N, these arrays contain $N \times N$ neurons. Combining these arrays, a controller for a 1-DOF birotor fixed to a frame was designed and executed on Intel's Loihi chip to control the roll

angle by inputting measurements of both the angle and the angular velocity. However, the influence of the derivative term in the response of the controller was not as important as for the proportional and integral terms, thus explaining the oscillations observed at zero-roll angles. In [22], the output of an event-based attitude estimate based on the Hough-transform was combined with this controller to perform attitude control. For both implementations, the amount of necessary neurons for a single controller scales quadratically with the resolution. Limits on the amount of neurons imposed by neuromorphic processors were restricting the precision of the representation of error and command values. To allow for smooth control the change in output command should have a large precision. Autonomous control with a cascaded-PID controller of a MAV in mid-air requires a bare-minimum of 6 PID controllers (but preferably more to include velocity control as well). Alternatively, in [79], a neuromorphic PID was designed to calculate the three terms of the controller using rate-coded signals, and further tested on-board the Loihi chip. Interestingly, rate-coding allowed to improve the derivative term by means of a set of different time scales for the synapses. However, such a solution remains hard to implement on the Loihi as the chip currently does not support high synaptic time constants.

These pioneer studies demonstrated the feasibility of using neuromorphic PIDs to control robots using neuromorphic processors such as the Loihi. Yet, having such systems working online to control a free-flying robot remains a great challenge, especially in the context of aerial vehicles where the robustness, reliability and execution speed of controllers is crucial. In this work, a different method of performing the mathematical adding (and subtracting) operations is suggested that reduces the number of necessary neurons by an order of magnitude as compared to the proposed method in [26], while also allowing for a non-linear distribution of the position-coded spikes that represent both the input, error and output floating point values. This simultaneously allows for full control of a MAV with a higher precision while keeping a smaller distribution, and thus more precision of the control output around zero. Furthermore, we successfully demonstrate the performance of our neuromorphic PID controller running on-board the Loihi neuromorphic chip for the altitude control of a free-flying MAV.

4.2. Methods

In the following, we introduce the standard cascaded PID controller and the neuromorphic PID (N-PID) for a quadrotor. The cascaded PID is hereafter considered as the baseline for comparison with the neuromorphic PID.

4.2.1. Cascaded PID controller

The cascaded PID controller used in quadrotors consists of multiple interconnected PIDs that produce rotor-speed commands for all four rotors. The design of the controller can be seen in Fig. 4.2, where each PID block represents a set of multiple PID controllers running in parallel to process the commands for the three axes (x, y, z). The continuous-time definition of a given PID controller is provided by the following set of equations:

$$\begin{cases} u(t) = K_P\Big(e(t) + \dfrac{1}{T_I}\displaystyle\int_0^t e(\tau)d\tau + T_D\dfrac{de(t)}{dt}\Big) \\ e(t) = r(t) - y(t) \end{cases} \tag{4.1}$$

where $u(t)$ is the control signal at time t, K_P the proportional gain, T_I and T_D the integral and derivative time-constants respectively and $e(t)$ the error signal between the target $r(t)$ and the measurement $y(t)$. This can be transformed into a discrete-time PID controller where the signals are defined as follows:

$$\begin{cases} e_k = r_k - y_k \\ i_k = i_{k-1} + e_k\Delta t \\ d_k = \dfrac{e_k - e_{k-1}}{\Delta t} \\ u_k = K_P e_k + \dfrac{K_P}{T_I} i_k + K_P T_D d_k \end{cases} \tag{4.2}$$

As illustrated in Fig. 4.2, the inputs to the first layer of the controller are position set-

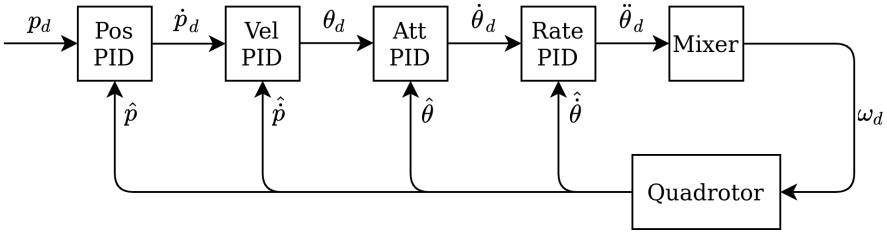


Figure 4.2: Cascaded PID for multirotor control where p is the position (x, y, and z) and θ are attitudes (θ, ϕ, and ψ), a dot ($\dot{a}$) represents the time-derivative whereas a hat ($\hat{a}$) stands for the measurement. ω_d are desired rotor speeds, calculated in the control mixer from the torque/thrust values.

points, which produce velocity set-points and so on. The outputs of the last PID are the thrust and torque commands that are translated into rotor speed commands in the control mixer based on the model parameters of the quadrotor.

4.2.2. Neuromorphic PID

The neuromorphic PID, referred to as the N-PID, is achieved by using populations of neurons to perform mathematical operations (i.e., additions and subtractions) necessary for computing the output command of the controller.

Input neurons population

We propose to use a set of N input neurons to encode the floating-point values using a standard position-coding scheme: each neuron will fire whenever the input value falls within a predefined range. The distribution of those sensitivity ranges can either be uniform, meaning that all neurons will have the same sensitivity, or non-uniform. In the latest, for instance, a quadratic distribution can be used to increase the neurons' sensitivity around a certain point of interest. Using such non-uniform, arbitrary distributions opens the opportunity for more accurate control around a certain set-point. To encode a floating-point value as a position-coded value, the distance from the floating-point value to all the N values represented by the encoded distribution is calculated. The neuron for giving the lowest error will then fire, while the other neurons will stay inactive (*winner-takes-all*).

Aggregate population layer

In the proposed N-PID, a population of neurons performs either an addition or a subtraction with two layers (Fig. 4.3): an *aggregate* layer performing the mathematical operation, then followed by a *reduce* layer that implements a position-coded winner-takes-all. The aggregate layer consists of two sub-populations of neurons, one for positive and one for negative outputs . The aggregate layer is densely connected to a set of input layers. For instance, if we aim at adding two input signals, there will be two populations of input neurons as represented in Fig. 4.3. To ensure the mathematical operation, the corresponding synaptic weights are linearly proportional to the inputs. However, connections to the negative sub-layer in the aggregate layer are multiplied by -1. This way, the negative values are represented by their absolute and the use of negative thresholds is bypassed. The thresholds of the aggregate neurons are proportional to the absolute of the values in the output range. This way, all neurons with a threshold lower than the sum of the input values in the correct sub-population will fire.

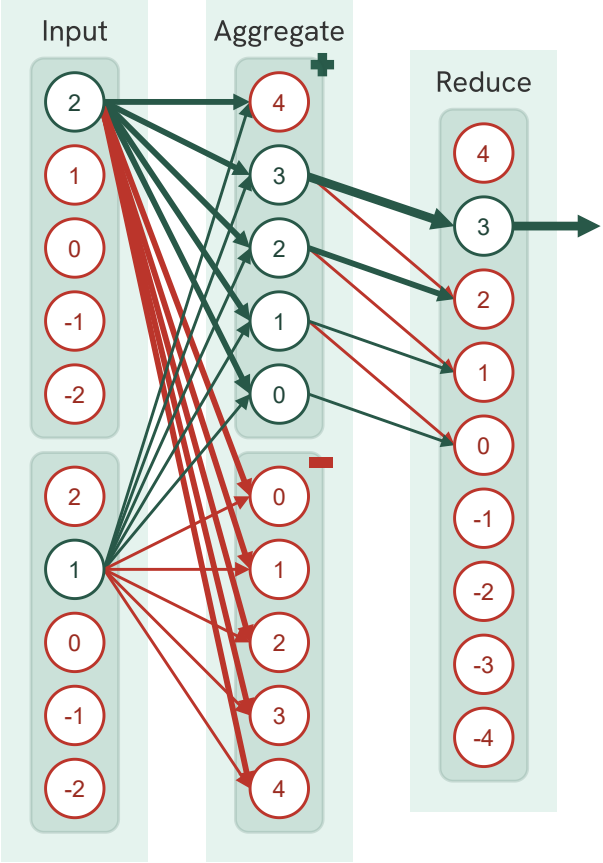


Figure 4.3: Adder neuron population for two input ranges of [-2, -1, 0, 1, 2] and an output of [-4, -3, ... , 3, 4]. This example shows the addition of the values 2 and 1. The green and red arrows represent excitatory and inhibitory synapses respectively. The membrane potential of all neurons in the positive group in the aggregate layer (upper five neurons) sums to 3, and in the negative group (lower five neurons) to -3. All neurons in the positive group (with thresholds indicated by the values inside the neurons) will now fire and in the reduce layer only the neuron representing the value 3 will fire.

Reduce population layer

The connection between neurons in the aggregate layer and neurons in the reduce layer is designed as follows (Fig. 4.3): first, an excitatory synapse connects the two neurons representing the same value, and second, an inhibitory synapse connects the same input neuron (aggregate) to the adjacent neuron in the reduce layer. Only the highest firing neuron in the aggregate layer will spike in the reduce

layer. Changing from addition to subtraction simply requires flipping the sign of the weights of the input to be subtracted. This brings the total amount of neurons necessary for performing an addition $2N + 1$ where N is the resolution.

Combining into a single N-PID unit

By combining these populations of neurons performing additions and subtractions, the N-PID can be implemented, as shown in Fig. 4.4. Since this design requires three of the previously described neuron populations (cf. Fig 4.3), the total amount of neurons necessary for the entire N-PID is $6N + 3$ if all sub-units have the same resolution, and excluding the input neurons ensuring the position encoding of the information.

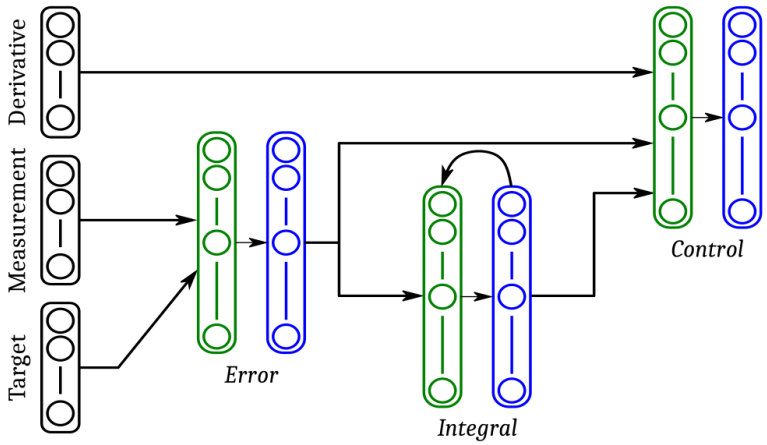


Figure 4.4: Overall structure of a single PID unit as implemented on the neuromorphic chip. Green layers stand for aggregate populations of neurons, and blue layers represent the reduce populations (cf. Fig. 4.3).

Integral wind-up

Integral wind-up is a well-known problem in PID design [80]. By taking the integral over time, error accumulates and the system might experience overshoot. One of two integral wind-up solutions is to bound the size of the integral, so it will not grow too large. Because of the nature of position-coding, this is already inherent to our N-PID. Another solution is to add a decay factor to the integral, ”forgetting” errors that lay further in the past. This can be implemented in this N-PID by multiplying the weights of the previous integral signal by a decay factor, that ensures faster

decay of the integral term when the set-point is crossed and will be used in our experiments.

4.2.3. Simulation setup

Both the cascaded PID and the N-PID were tested in simulation. An AscTec Hummingbird quadrotor was simulated in RotorS, a ROS/Gazebo physics simulator that enables quick testing of ROS packages before implementing in a real system [27]. State estimation was achieved by utilizing the MSF framework, providing 6-DOF state estimates based on an Extended Kalman Filter (EKF) [81]. The simulator runs on a Dell XPS 17 (Intel i7-10750H 12-core 2.6GHz CPU and 16GB RAM) running Ubuntu 18.04 LTS. The cascaded PID was implemented as a ROS package in Python (*version 3.6*), while the N-PID has been implemented for height control.

To make sure that the N-PID can run on the Loihi chip, a transformation from floating point weights is necessary. Loihi weights supports 8-bit resolution, with 1 bit reserved for the sign. This means the weights have to be between -256 and 254, with steps of 2. Due to the discretization inherent to this type of network, the first-order derivative suffers largely from jumps in the input bins, especially with lower precision. To counter that, the error derivative is directly fed to the control-layer of the N-PID.

Besides, it is essential to have an accurate error term in the controller of an unstable non-linear system such as a quadrotor. This controller receives set-points and state estimates and produces thrust and torque commands. These commands are then transformed into rotor velocity commands, which are then sent to the quadrotor model implemented in RotorS. In this simulation, the step responses of the quadrotor to a change in the height set-point are simulated between 1 and 3 meters.

4.2.4. Hardware setup

The performance of the neuromorphic controller in the real-world has been assessed. The controller was implemented to control the height of a quadrotor. The proposed architecture features a custom made 5-inch MAV equipped with (i) the Kakute F7 flight controller running the iNav 2.6.0 open-source firmware equiped with a TFmini Micro-Lidar for altitude measurements, (ii) the Intel UP board (1.92GHz 64bit Atom processor with 4GB RAM) running Ubuntu 18.04 LTS, and (iii) the Intel Kapoho Bay neuromorphic computing unit incorporating two Loihi chips (Fig. 4.5). The entire project has been developed within the ROS

framework (Robot Operating System) running onboard the UP board. The communication between the flight controller and the UP board is ensured by serial communication utilizing the MultiWii Serial Protocol (MSP). On the other side, the communication between the UP board and the Kapoho Bay is performed by means of Intel's NxSDK API [82] with ROS-compatible modules, and physically achieved via the USB interface available on the Kapoho Bay.

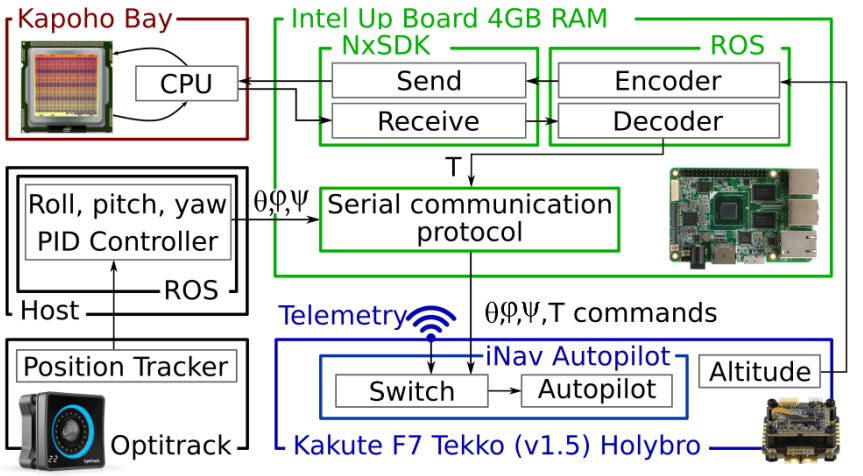


Figure 4.5: Overview of the hardware architecture of the MAV equipped with the Kapoho Bay. In the context of this study, the neuromorphic PID for height control runs on the Kapoho Bay (thrust T), while the roll (θ), pitch (ϕ) and yaw (ψ) controller is done on the host machine which communicates with the quadrotor via a wireless UDP protocol. In this graph, the colored boxes (Up board, Kakute, and Kapoho Bay) indicate indicate that these elements are on-board the drone, while the black boxes (host and OptiTrack) remain outside the drone.

The altitude sensor (Micro-LiDAR) provides height measurements sent to the UP board, where the N-PID calculates the desired thrust and sends this information back to the flight controller. The attitude commands are sent from a base station based on the position measurements provided by the Optitrack motion tracking system.

4.3. Results

4.3.1. Simulation results

The step response of the N-PID controller was first compared to the standard PID in the context of altitude control of a quadrotor for varying altitude set-points

ranging from 1.0 to 3.0 meters with a step of 0.5 meters. The N-PID provides the thrust-offset T from hover-thrust. An initial estimate of hover-thrust is obtained by multiplying the weight of the quadrotor with $g = 9.81$, further adjusted by hand to account for uncertainties in the model. The gains of both PIDs were tuned by hand ($K_P = 0.87$, $T_I = 0.17$, $T_D = 2.76$). The target and measured value ranges were both chosen within $[0, ..., 4]$ meters and the derivative of the error ranged from -0.5 to 0.5 meters. The range of the output control T was chosen within $r = [-1.25, ..., 1.25]$ (offset for the hover-thrust).

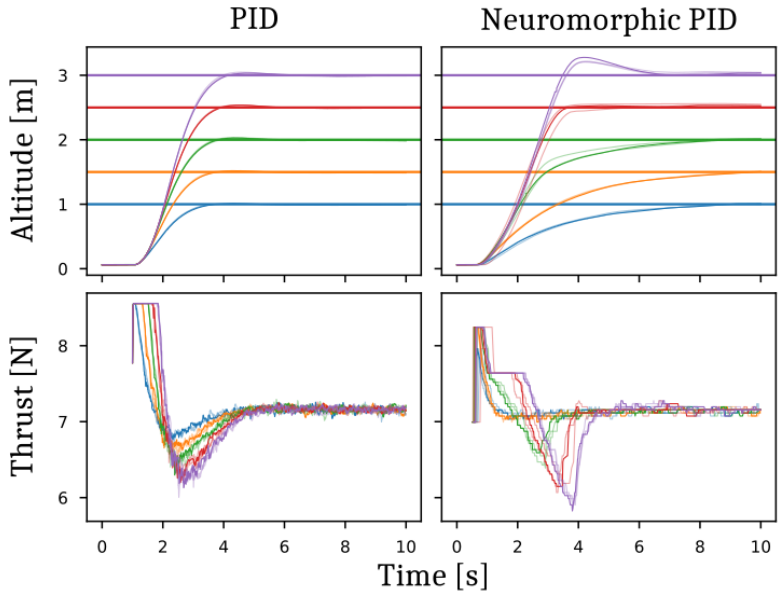


Figure 4.6: Step response of the N-PID with a population of 151 output neurons, compared to the standard PID for different height set-points (1.0, 1.5, 2.0, 2.5, and 3.0 meters). These results were obtained in simulation.

In Fig. 4.6, we provide the step response of the conventional PID and the N-PID, provided with an overall precision of 0.017N (151 output neurons uniformly distributed). For each altitude set-point we ran 5 distinct simulations. Although both systems manage to reach the target, we notice that the N-PID exhibits a slower dynamic than its conventional counterpart, with higher overshoot for high altitude set-points. Where the conventional PID has a similar settling behaviour for all heights, the N-PID has undershoot at lower heights, but overshoot at higher ones.

We further investigated the effect of the control precision (i.e., the number N of

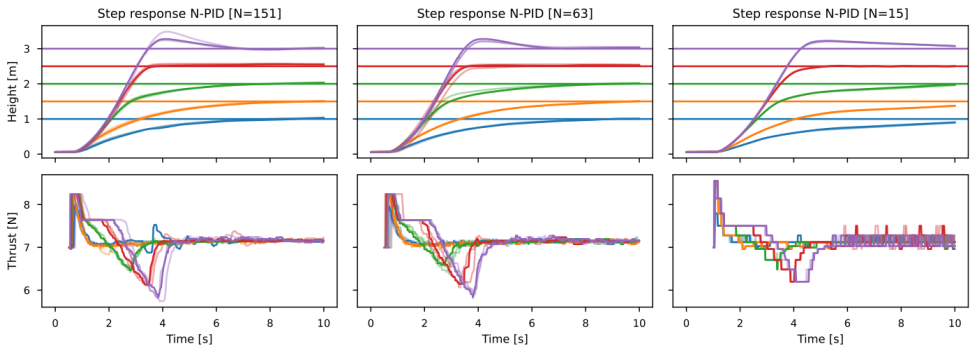


Figure 4.7: Step responses of the simulated neuromorphic PID (N-PID) applied to height control of the MAV and for varying precision (i.e., 151, 63, and 15 neurons). The target altitudes are 1.0, 1.5, 2.0, 2.5 and 3.0 meters.

output neurons) on the step response of the N-PID. The results are shown in Fig. 4.7 for a number of output neurons $N \in [151, 63, 15]$. For the two first tests ($N = 151$ and $N = 63$), the population of output neurons obeys a linear distribution, thus ensuring all neurons contribute to the exact same control resolution. Results show that the size of the population of output neurons does not have a significant impact on the overall dynamics of the N-PID.

In case of a population of only 15 neurons, the control range was changed to a quadratic distribution as follows: $\text{sign}(r) \cdot r^2$. This way, the control around hover-thrust is more precise than far from it, hence compensating for the very low resolution that a uniformly distributed population of 15 neurons would have led to (0.17N). In simulation, we observed that the N-PID does not exactly reach the set-point because of the discretization in the target (and measured) position. This means that, because the range of sensitivity of each neuron is quite large, the error has a higher chance to be equal to zero while the target altitude has not been reached yet. Nevertheless, the results obtained with the N-PID endowed with a population of only 15 neurons to represent the control command with a quadratic sensitivity distribution are promising and suggest that this setup should be reliable enough to be further applied on the MAV.

As the final goal is to have the N-PID running on-board the Loihi chip mounted on the MAV, a thorough comparison of the spiking activity of the network has been performed to compare the simulated N-PID to the network implemented on the Loihi. This analysis resulted in a perfect match between the two setups, thus ensuring that the switch from conventional to neuromorphic hardware is not increasing the reality gap. An overview of the spiking activity recorded on the

Loihi is provided in Fig. 4.8 for a target altitude of 1.5 meters. In order to simplify the visualization, only the output layers of sub-units (error, integral, motor command) are shown. The sparsity of the spiking activity contributes to the parsimony of the proposed N-PID. The average execution time per time step of the entire network has been assessed on the Loihi, showing a total of $2\mu s$ per time-step, thus resulting in an average update frequency of 500kHz, which is far beyond what we need for online applications.

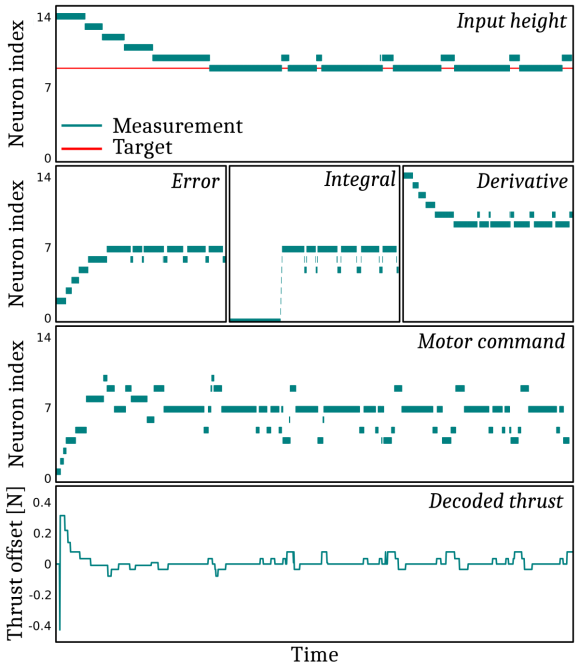


Figure 4.8: Overview of the spiking activity on-board the Loihi chip running the N-PID height controller in simulation (offline data). In this case, the target altitude is 1.5 meters. The thrust offset corresponds to the offset to be added to the hovering thrust command.

4.3.2. Real-world results

In the following, we implemented the N-PID for altitude control on the Kapoho Bay neuromorphic device mounted on-board a MAV. Based on the results obtained in simulation, we decided to test the small network with a population of 15 output neurons with a quadratic distribution of the control precision. Therefore, the N-PID features only 93 neurons ($6N + 3$), plus 45 input neurons for the encoding of the altitude measurement and target, along with the derivative of the error.

Two different altitude set-points were tested (1.0m and 1.5m) over 5 trials each. The altitude ground truth over time was given by the OptiTrack motion capture system, and the results are shown in Fig. 4.9. The shaded colored areas show the target bins, caused by the precision of the altitude discretization. Inside the shaded area, the error between the measured and target altitude is zero, which means the MAV is flying at hover-thrust. For some flights it can be seen that the drone is not able to maintain altitude. On the real drone, the hover thrust PWM level depends largely on the battery level. Even though it is visible that the integrator is able to compensate for a voltage drop over multiple runs, the integral output will be saturated and the MAV will lose altitude with a low battery. During the tests, the control loop update frequency of the system was set to 70Hz to be consistent with the height sensor.

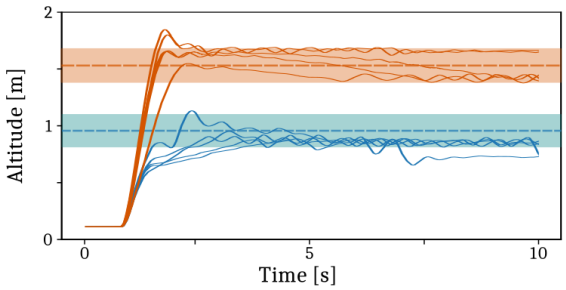


Figure 4.9: Real-world tests for take-off altitude control with neuromorphic hardware for $N = 15$. Different set-points are shown in different colors. The shaded bands around the set-point represent the bin around the set-point altitude

In Fig. 4.10 the I/O spiking activity of the N-PID running on the neuromorphic chip is shown for one of the tests performed with the MAV. Compared to the simulation results, there is more noise around the thrust command. This is caused by the derivative that is based on the measured altitude and has a resolution of 1 cm.

4.4. Conclusion

We presented a neuromorphic implementation of the PID to control the altitude of a MAV equipped with the Loihi neuromorphic processor. Using a very low number of neurons, the network demonstrated its capability of achieving robust and stable control, with the potential of reaching extremely high control loop frequencies. The proposed neural network does not require any training as its circuitry inherently captures the desired dynamics – only the weights of the synapses need to be tuned to fit with the aerodynamic properties of the MAV. In

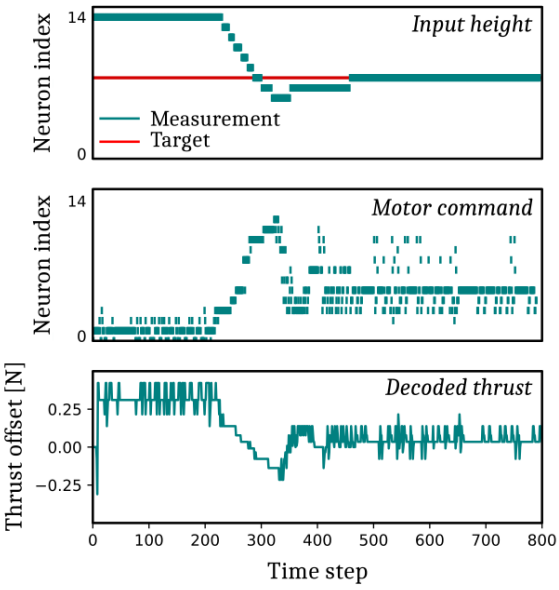


Figure 4.10: Overview of the Loihi input/output spike activity during an online test with the MAV (target altitude: 2.0m), as well as the decoded thrust offset (in Newtons). The controller generates a slight overshoot for high targets, but quickly converges to the desired set-point.

practice, the tuning of the input, error and control ranges, can be difficult for unstable systems with a low amount of neurons. Here, using non-linear distributions for these ranges has proved to mimic the response of a controller with a larger amount of neurons and is very promising for attitude control.

In both Stagsted et al. [26] and Zaidel et al. [79], it was observed that implementing a proper derivative action on neuromorphic hardware is difficult and remains an open problem. Future improvements of this algorithms include a fully neuromorphic implementation of the N-PID for controlling all degrees of freedom of the MAV, including attitude commands. This requires multiple PIDs to run in parallel and series on the neuromorphic hardware.

Having a neuromorphic controller like a PID that is easily implementable on available hardware contributes to closing the neuromorphic loop for control in robotics. These controllers can be easily implemented in pipelines making use of event-based algorithms.

Lastly, the very high execution frequency of the N-PID on the Loihi chip (around 500kHz) strengthens the fact that conventional hardware (including communication protocols) must be improved to meet the promise of neuromorphic computing to

target very fast control in tasks such as drone racing and aggressive maneuvers with MAVs.

Supplementary information

The code and data collected during this study are publicly available online at the following address: `https://github.com/tudelft/neuro_pid`.

Acknowledgements

This material is based upon work supported by the Air Force Office of Scientific Research under award number FA8655-20-1-7044. This work has also received funding from the ECSEL Joint Undertaking (JU) under grant agreement No. 826610. The JU receives support from the European Union's Horizon 2020 research and innovation program and Spain, Austria, Belgium, Czech Republic, France, Italy, Latvia, Netherlands.

Link to paper:

5

Threshold adaptation facilitates integration

As previously discussed in this thesis, it is currently still challenging to replicate even basic low-level controllers such as proportional-integral-derivative (PID) controllers. Specifically, it is difficult to incorporate the integral and derivative parts. To address this problem, we propose a neuromorphic controller that incorporates proportional, integral, and derivative pathways during learning. Our approach includes a novel input threshold adaptation mechanism for the integral pathway. This Input-Weighted Threshold Adaptation (IWTA) introduces an additional weight per synaptic connection, which is used to adapt the threshold of the post-synaptic neuron. We tackle the derivative term by employing neurons with different time constants. We first analyze the performance and limits of the proposed mechanisms and then put our controller to the test by implementing it on a microcontroller connected to the open-source tiny Crazyflie quadrotor, replacing the innermost rate controller. We demonstrate the stability of our bio-inspired algorithm with flights in the presence of disturbances. Integration is an important part of any temporal task, so the proposed Input-Weighted Threshold Adaptation (IWTA) mechanism may have implications well beyond control tasks.

Parts of this chapter have been published in The Proceedings of the International Conference on Neuromorphic Systems (ICONS) 2023 [83]

5.1. Introduction

Autonomous drones are envisaged for a wide range of applications [84]. Many of these applications require a high computational capability, which enables them to accomplish tasks solely based on data acquired from their onboard sensors, such as cameras and GPS-sensors. Currently, this is out of reach for many drones, since they are very limited in terms of size, weight, and processing [85]. Deep neural networks require heavy, power-hungry processors. That is why there is a surge of interest in bio-inspired, neuromorphic processing, which carries the promise of low-latency, energy-efficient processing of deep neural networks [86]. Although there is a lot of focus on complex neural networks for high-level visual perception [87], a fully neuromorphic solution needs to encompass low-level control [88]. Remarkably, it is currently still highly challenging to replicate even simple low-level controllers such as PID controllers with spiking neural networks. An example of such a low-level controller is the fascinatingly elegant biological system that affects the haltere reflexes of the *drosophila melanogaster* [89].

Recently, an increasing amount of robotics research has been focused on

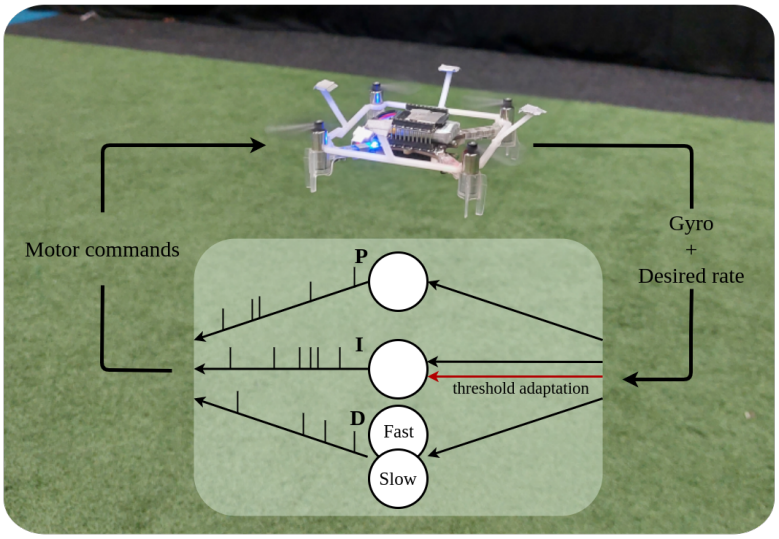


Figure 5.1: We propose a novel spiking neural network mechanism for realizing the integral term in a spiking PID controller and analyze the use of different time constants for the derivative term. For the integral term, we introduce Input-Weighted Threshold Adaptation, leading to a second weight per synapse. These mechanisms are demonstrated with onboard attitude rate control of a tiny Crazyflie drone.

developing spiking neural networks (SNNs) for control. Specifically for controlling flying robots, examples include Clawson *et al.* [90], which uses reward-modulated synaptic plasticity to track a Linear-Quadratic Regulator (LQR) for a flapping wing drone. In Qiu *et al.* [91], a neuro-evolution strategy is utilized to learn a controller for a drone and is shown to outperform a PID in simulation. Closer to our work are studies that mimic the behavior of conventional controllers with SNNs. Among those, the benefits of a spiking end-to-end control pipeline are especially clear in Vitale *et al.* [22]. In this work, the rotation of a bi-rotor was controlled by combining a neuromorphic implementation of the Hough transform with a population-coded spiking implementation of the conventional PID controller. They showed that due to the high update rates and asynchronous data flow from the event camera and neuromorphic chip, much faster responses could be obtained than with a conventional control setup. The accuracy of this network scaled quadratically with the number of neurons, putting a limit on the resolution. In our previous work, the complexity of such a PID network was reduced to make it scale linearly with the number of neurons, and the network was used to control the altitude of a free-flying drone in a real-world test [66]. However, the integral and derivative paths in both these works showed clear limitations, imposed by the number of neurons used to represent the signals.

Besides population coding, also rate coding was used to recreate conventional control. For example, in Zaidel *et al.* [79] the joints of a 6-DOF robotic arm were controlled by a rate-encoded PID controller, creating the integral pathway by having self-recurrent connections and the derivative by adopting different time constants. Despite using this self-recurrency in the integral pathway, there still remained a steady-state error, showing the incapability of error integration over time.

In all these examples the seemingly simple tasks of calculating the integral and derivative over time have proven to be difficult for the current-based Leaky Integrate-and-Fire (LIF) model. This may be due to the simplicity of the LIF neuron used in most robotics research. Popular for its simplicity, it fails to deal with the phasic-tonic response exhibited by biological neurons [92]. Lastly, although all these works show clear steps toward a full end-to-end neuromorphic pipeline, none of them has demonstrated the ability to control the lowermost loop of a real, free-flying drone.

We present a neuromorphic controller that can more closely mimic PID controllers. We make the following contributions: (1) We introduce Input-Weighted Threshold Adaptation (IWTA) to achieve more accurate integration. (2) We systematically study the capabilities and limitations of neurons

with slow and fast time constants for obtaining the derivative term. (3) We analyze the advantages and limitations of the introduced mechanisms. (4) We demonstrate the novel neuromorphic controller with the onboard attitude rate control of a tiny (≈30 gram) flying Crazyflie robot (shown in Figure 5.1). Using only 320 neurons and 800 synapses per PID line, it is designed to take advantage of the unique characteristics of neuromorphic computing, such as high processing speed and fault tolerance, while maintaining the promise of low power consumption when implemented in specialized hardware. The network is automatically trained to closely track the output of a conventional PID using backpropagation-through-time (BPTT), removing the necessity for manual tuning of network parameters.

5.2. Methods

The entire SNN consists of multiple groups of LIF neurons, each resembling one of the separate parts of a conventional PID controller. All groups consist of an encoding layer, transforming floating point inputs to rate encoded spike-trains representing positive and negative values. These spikes propagate to neurons that respond proportionally to the input, to the accumulated signal over time or to the rate of change of the input. These independent systems are discussed in detail below.

5.2.1. Encoding - floating points to spikes

For our network, a rate encoding scheme has been chosen that has separate channels for positive and negative values. To ensure a certain spiking frequency at an input stimulus, encoding is done according to two (symmetrical) tuning curves. These tuning curves represent the spike probability of an encoding neuron to a given stimulus, and by comparing this to a random-generated number, either a spike (1) or not (0) is produced by the neuron. Although more complex functions can be chosen for these tuning curves, in this work a linear relation between spiking frequency and stimulus was chosen. This linear relation between input $i(t)$ and output spike probability $P(s(t))$ at time t is dictated by the

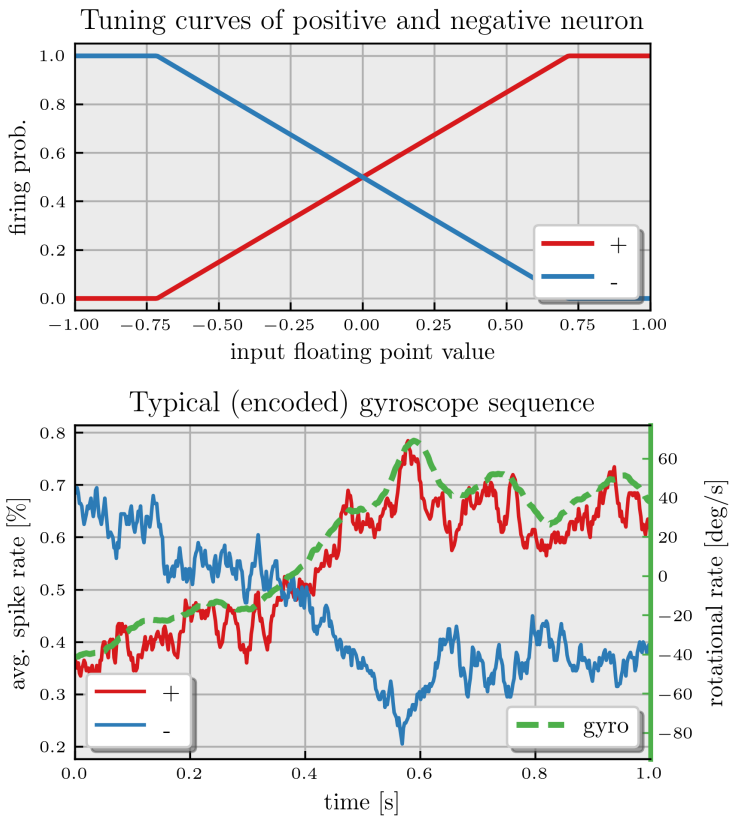


Figure 5.2: Overview of the encoding layer. The top graph shows the firing probability for a certain stimulus for positive and negative encoding neurons. The bottom graph depicts a typical sequence of encoded gyroscope measurements. The average spike rate of the encoded positive and negative spikes is shown in blue and red and the input sequence is shown in dashed green.

parameters α and β as follows:

$$P(s(t)) = \begin{cases} 0 & r\beta i(t) + \alpha \leq 0 \\ r\beta i(t) + \alpha & 0 \leq r\beta i(t) + \alpha \leq 1 \\ 1 & r\beta i(t) + \alpha \geq 1 \end{cases} \tag{5.1}$$

$$r = \begin{cases} 1 & \text{positive neuron} \\ -1 & \text{negative neuron} \end{cases} \tag{5.2}$$

The value of r depends on whether the encoding neuron is positive or negative. This is visualized in Figure 5.2 where the set of tuning curves used in this work and the resulting output spikes are shown for a typical input sequence measured with gyroscopes.

5.2.2. Proportional - steering towards setpoint

The output of the proportional layer should drive the system from its current state to a setpoint by responding linearly to the error. In this network, this is done by feeding the output of an encoding group, as described above, to two current-based Leaky Integrate-and-Fire (LIF) neurons, again each representing either positive or negative control commands (see Section S5.5.1 for the model used). The spikes from this layer are sent to an output leaky integrator, acting as an exponential filter to smooth the response. To ensure a balanced response, the synaptic weights in the network must be symmetrical for the positive and negative inputs. By using more than one group, the stochastic effects induced by the encoding layer can be reduced and the accuracy increased.

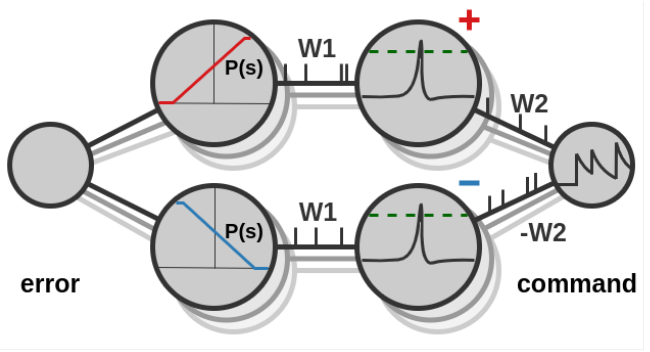


Figure 5.3: Structure of the proportional neuron groups. The value is encoded into ”positive” and ”negative” spikes and sent via symmetrical weights to a layer of Leaky Integrate and Fire neurons. The spikes emitted by these neurons are sent to a Leaky Integrate output neuron, resulting in an exponential filtered output.

5.2.3. Integral - ensuring zero steady-state error

The integral neurons should remove the steady-state error, unresolved by the proportional control. Initially, one may think that LIF neurons could integrate by

setting the decay parameter to one (no decay). However, as soon as a spike occurs the integrated membrane potential is reset. Hence, if the membrane potential is not read out directly and the integrator has to be encoded by spikes, a different mechanism is required. We propose a different solution in this work inspired by threshold adaptation mechanisms and the modulatory effects certain receptors exhibit. There is a synaptic connection between the positive and negative encoding neurons to both neurons, as opposed to the proportional connections where there is only a connection between the positive neurons on the one hand and between the negative neurons on the other. Besides the synaptic weights, there is also a signal flowing from the encoding neurons to the thresholds of the neurons in the integration layer, increasing or decreasing the threshold based on the sign. Previous work has studied various mechanisms for adapting the threshold based on inputs, often to prevent spike saturation. For example, the regular Adaptive LIF (ALIF), adapts its threshold based on its own spike activity [93]. As a variation of the ALIF, Paredes-Vallés *et al.* [94] proposes to deduct the pre-synaptic spike trace from incoming spikes, to discern features under varying input statistics, such as the per-pixel firing rate of event camera data. In our method, however, the threshold is regulated based on weighing the incoming activity. A spike in the *positive* integration neuron coming from the *positive* encoding neuron decreases the threshold with a weight of θ_{add} (therefore increasing the spiking rate), while the *negative* encoding neuron causes an increase with θ_{add} (thus decreasing the spiking rate), and vice-versa for the *negative* integration neuron. This results in the following update rule for the threshold:

$$\theta^{\text{thr}}(t+1) = \theta^{\text{thr}}(t) + \theta_{\text{add}}(s_{-}(t) - s_{+}(t)), \tag{5.3}$$

$$\theta_i^{\text{thr}}(t+1) = \tau^{\text{thr}}\theta_i^{\text{thr}}(t) + w^{\text{thr}}s_i(t) \tag{5.4}$$

$$\theta_i^{\text{thr}}(t+1) = \tau^{\text{thr}}\theta_i^{\text{thr}}(t) + \sum_{j=0}^{N} w_j^{\text{thr}}s_j(t) \tag{5.5}$$

with $s_{-}(t)$ and $s_{+}(t)$ the negative and positive incoming spikes, respectively. Now, the encoding neurons act as a constant driving synaptic signal to maintain a certain activity in the integration layer, while the actual spiking rate is governed by the variation in the threshold. A common problem with PID regulators is integral windup, where actuator saturation or large changes in setpoint might lead to large amounts of accumulated error [95]. In our LIF model, we solve this by limiting the amount of change in the threshold, keeping the threshold in the range

of $[0, 2\theta^{\text{thr}}]$ where θ^{thr} is the base threshold. If the threshold is zero, the integrator will spike with the maximal spiking frequency, which is determined by the time step and refractory period. It could be noted that multiple changes to this model

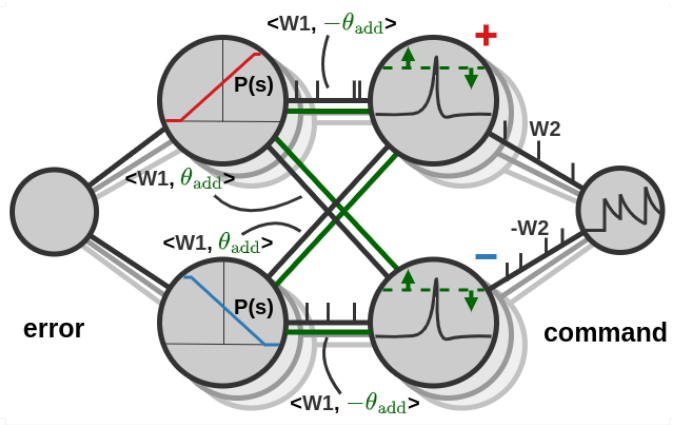


Figure 5.4: Structure of the integral neuron groups, bearing the Input-Weighted Threshold Adaptation (IWTA) mechanism. Next to the regular synapses, additional connections have been added that adapt the threshold of the LIF neurons according to their weights.

are imaginable. For instance, if a decay term is added to the threshold it more closely resembles the ALIF, where the threshold converges back to a base value. In biology, this kind of input adaptation is similar to certain neurotransmitters with modulatory effects [96]. Also, every input group now uses only a single positive and negative encoding neuron, with one update parameter θ_{add}. One could imagine using a larger group of encoding neurons, separate update weights θ_{add} per input connection and including these weights in the training procedure. In this work, this remains unexplored since it focused on the task of integrating errors for which the proposed set of connections was sufficient.

5.2.4. Derivative - decreasing overshoot

The derivative action should be proportional to the change over time, countering any potential overshoot. To obtain this, a similar structure to the proportional groups is used (as in Figure 5.3), but now two of these groups are used in unison, instead of one. One of these groups has very fast time constants (higher weights, but faster decay), allowing it to react quickly proportionally to the input. The other has slower time constants (lower weights, but slow decay), resulting in an output that is a slightly delayed version of the input. By taking the difference

between these two groups, we get a measure of the change over time which can be used as the derivative term in our PID controller. Using multiple of these derivative groups again increases the accuracy of the overall estimate. For the derivative, this is especially important, as the derivative is usually already quite noisy due to noisy sensor measurements.

5.2.5. Training and tuning of the network

Since the network has a substantial number of parameters that all influence the performance of the controller (such as synaptic weights, decay parameters and encoding parameters) and the parameters all depend on the time constants and gains of a specific controller, manual tuning is undesirable. Therefore, it was chosen to use the architecture of the network as described above as a starting point and fine-tune the parameters using Backpropagation-Through-Time (BPTT). Because the LIF threshold function is non-differentiable, it was chosen to apply a surrogate gradient in the backwards-pass of BPTT [97]. Specifically, the surrogate gradient used in this work is the derivative of the arc-tangent as was proposed in [98].

To force a response that is close to that of the target, the Mean Squared Error (MSE) was used as the dominant term in the cost function. Since during training, especially the derivative term was very sensitive to converging to local minima, it was chosen to add a second cost term for the derivative based on the Pearson correlation coefficient $\rho(x, \hat{x})$ [99]. Since we have a minimization problem and the perfect coefficient ρ is 1, we use $1 - \rho(x, \hat{x})$ as the cost, further referring to it as the Pearson loss. For derivative control, it is very important that the control action is at least of the correct *sign* because failing so might mean instabilities can arise. The Pearson loss promotes a linear relationship between both inputs and therefore supports the network to produce the correct sign. Finally, the parameters of the network need to stay within certain bounds, decay parameters for instance cannot be larger than 1 or smaller than 0. To force parameters to stay within these bounds, a linear exterior penalty function $\mathsf{p}_{\mathsf{err}}(\phi)$ is added to the cost function equal to the distance to the boundary. This results in the following cost function used in the BPTT training algorithm:

$$J(\phi) = \mathsf{MSE}(x, \hat{x}) + (1 - \rho(x, \hat{x})) + \mathsf{p}_{\mathsf{err}}(\phi). \tag{5.6}$$

In this cost function, x and $\hat{x}$ are the measured- and estimated values respectively, $1 - \rho(x, \hat{x})$ the Pearson loss. $\mathsf{p}_{\mathsf{err}}(\phi)$ is the error based on the parameters ϕ, which is zero for all values of ϕ inside their specific range but of size $|\phi|$ for those outside.

A table of all parameters included in the training, together with their valid ranges, can be found in the supplementary information.

The data used for training was accumulated by logging the PID values of the regular Crazyflie controller on an onboard μSD card during manual flight. Care was taken to excite the system enough to gather a broad range of possible values a controller might encounter.

5.3. Analysis SNN controller

First, we look at the suitability of the multiple time constants for differentiation and afterward, we evaluate the IWTA mechanism for integral control.

5.3.1. Derivative

To assess the ability of a network of LIFs with different time constants to estimate the derivative we start by looking at the response of both the fast and slow groups to an illustrational gyroscope sequence, obtained with the Crazyflie, after training. As can be seen in Figure 5.5, the average spiking rate of the slow groups is approximately a delayed version of the fast groups. By subtracting the delayed version we obtain a result similar to first-order backward difference, usually used in robotics to calculate the derivative of sensor data. We noticed, however, that a particular set of time constants fits chiefly to the data it was trained upon. To further investigate this behavior, the response to sine waves with different frequencies was studied. Figure 5.6 shows the MSE and Pearson loss for a range of sine waves with different frequencies after training on two sets of frequencies. The network was trained on a smaller range of sine waves between 5 and 7Hz, and also on a much wider range of 2 to 10Hz. Afterwards, the response to the entire frequency range was compared for both trained networks. Although the network can accurately determine the derivative for the middle frequencies (B), it is too fast for lower frequencies (A), where the network in blue rises to its peak faster than the real derivative in red, and too slow for higher ones (C) where the network reaches its tipping point later. It is also visible that even with the larger range of input frequencies during training, the different time constants cannot correctly represent the entire band. Although the overall error gets reduced when training on the large band, the response for the lower and higher frequencies is still inaccurate. This suggests that a different mechanism would need to be introduced in order to obtain perfect differentiation independent of input frequency.

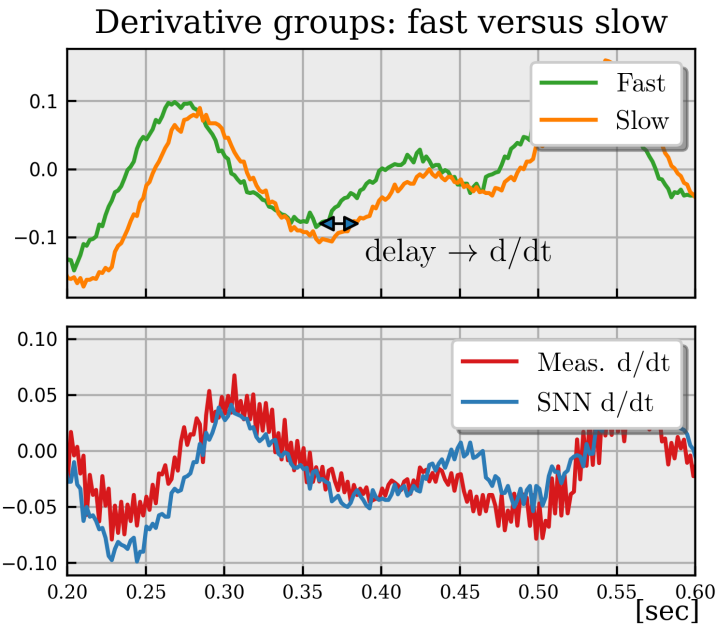


Figure 5.5: Example response to a common gyroscope input sequence recorded with the Crazyflie of both derivative groups. The top graph shows the average spiking rate of the fast and slow groups. As can be seen, the slow group is slightly delayed and the difference between both can be seen as a measure of the rate of change over time. In the bottom figure, this is visible as the derivative estimated with the SNN is compared to the measured derivative.

The importance of both cost terms in the loss function is also visible; the MSE cost for lower frequencies is relatively lower than the Pearson loss and for higher frequencies vice versa. Also, the amplitude of the SNN does not scale with the frequency, as is the case in the real derivative. This asserts the need for a reliable training procedure to tune the network to the application used, as was done in this work.

5.3.2. Integral

Furthermore, we have looked into the potential of the proposed Input-Weighted Threshold Adaptation (IWTA) mechanism. To present the importance of a functioning integrator, the classical control problem of the double integrator is used as an example. We have implemented the discrete-time equations of the

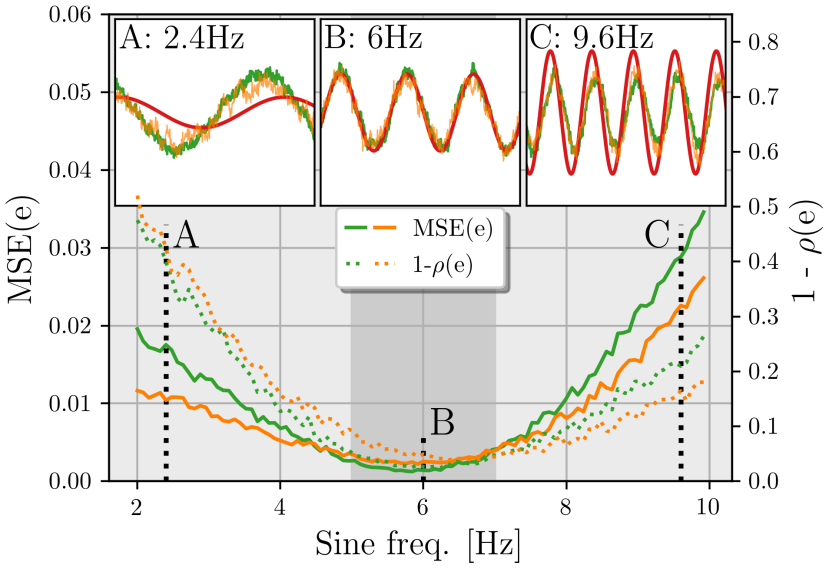


Figure 5.6: MSE and Pearson loss for a range of derivatives of sine waves after training on two sets of frequencies, 5-7Hz and 2-10Hz. Green lines correspond to the smaller range, and orange to the larger while the solid lines show MSE and dotted Pearson loss. The darker-colored area depicts the range on which the smaller set was trained. Three sub-figures are also included, demonstrating the response to a frequency A) below, B) inside and C) above the smaller training range with the SNN response in green and orange and real derivative in red.

double-integrator as

$$x(t+dt) = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} \frac{1}{2}dt^2 \\ dt \end{bmatrix} (u(t) - g), \tag{5.7}$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t), \tag{5.8}$$

where $u(t)$ is the control input and g is a constant input disturbance. In Figure 5.7 we show the response of this system to three different SNNs; the LIF SNN with only PD components, an SNN with PD components and an integrator group that has fully recurrent connections (as was used in [79]) and lastly our SNN with IWTA. The system starts from an initial state of $x0 = [0.3, 0.0]^\mathsf{T}$ and gets a setpoint 0.0. The constant disturbance force g was chosen to be 4.0. First, the SNN without any integrating mechanism settles at a steady-state error of -0.1. Second, the SNN with recurrent connections manages to reduce this steady-state offset, but not

remove it completely. Due to representation errors, the recurrent connections cannot perfectly describe the integrated error. This means that the accumulated error is either underrepresented or overrepresented. In the former case, the integrator leaks information each time step and thus never completely removes the steady-state error. The green lines in Figure 5.7 show this, since it clearly reduces the steady-state error, but does not remove it completely. In the latter, the feedback amplifies the error at each time step which makes the system unstable and leads to all integrator neurons spiking at each step. Finally, the network that uses IWTA reaches a zero steady-state error.

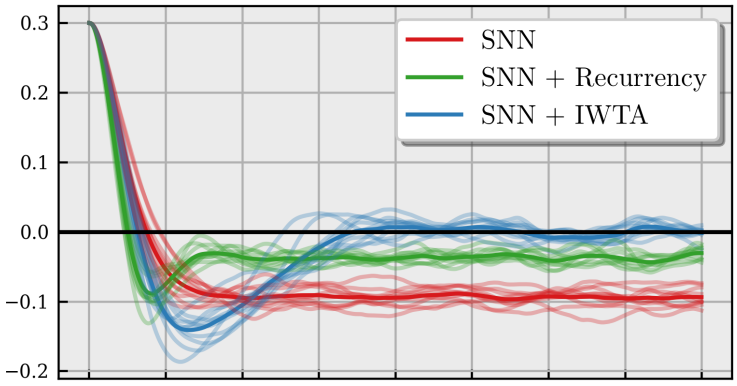


Figure 5.7: Comparison of three controllers to a setpoint-control task with an unknown input disturbance. The SNN with PD control only shows a constant offset from the setpoint after converging. The SNN with recurrency is clearly able to reduce the steady-state error, but due to representation error retains a small offset. The network equipped with Input-Weighted Threshold Adaptation effectively removes the steady-state error.

5.4. Real-world experiments

After the components are evaluated separately in simulation on toy problems, the use of the complete network was verified on a real-world problem.

5.4.1. Hardware implementation

To demonstrate the capabilities of our approach, we have implemented it as the lowest control layer of the tiny open-source quadrotor Crazyflie [100] (See

Figure 5.1). The Crazyflie was enriched with enough computational power by the development of a deck module based on a Teensy 4.0 development board allowing the SNN to run in real-time (500Hz) in C++ on the ARM Cortex-M7 microprocessor. For the real-world tests, two scenarios were discerned; 1) manual flight and 2) position control. For manual control the Crazyflie receives attitude (roll/pitch/yaw) commands from a (manually controlled) radio transmitter and the higher-level attitude controller transforms these into rate setpoints. In the case of position control, the Crazyflie receives position measurements obtained with a Motion Capture system along with position commands via radio and via the same high-level controller these are converted to rate setpoints. These setpoints are sent, along with the gyroscope measurements from the onboard Inertial Measurement Unit (IMU), via UART to the deck where the SNN controller is evaluated at 500Hz. The torque command outputs of the neural controller are in turn sent back to the Crazyflie via the same UART connection, where they are inserted directly in the motor mixer. The total take-off weight of the Crazyflie including the Teensy 4.0 is only 35 grams, allowing for approximately 5 minutes of flight time.

5.4.2. Flight tests

To demonstrate the capabilities of the network, a position-control test was performed along with manual flight. First, the Crazyflie was ordered to take off at its current position and after 2 seconds it was ordered to move to [0.5, 0.0] in 2 seconds. This test was repeated 10 times for both the SNN controller and the regular PID controller as a benchmark. The results of all these 10 tests are shown in Figure 5.8. In the top two plots, the response of the SNN controller is shown. Among all ten tests, the controller remained stable and was able to follow the setpoint. The trajectory followed is very similar to that of the regular PID controller. However, on the y-results, as well as the inset axes for x, it is visible that the SNN controller has slightly larger deviations around the setpoint. These can be caused by the stochasticity in the encoder. Since the input floating point value is reduced to a binary spiking input, the accuracy of the encoding is influenced by the encoding parameters but mostly by the number of input groups used. For this work, only 40 groups were chosen per P, I and D pathway for each control axis. Since the P and I terms require 2 neurons per group and the D term 4, this sums up to a total of 320 neurons per controller. This small number of neurons was chosen to showcase the possibilities of using small-scale systems for neuromorphic control. However, increasing the number of input groups would

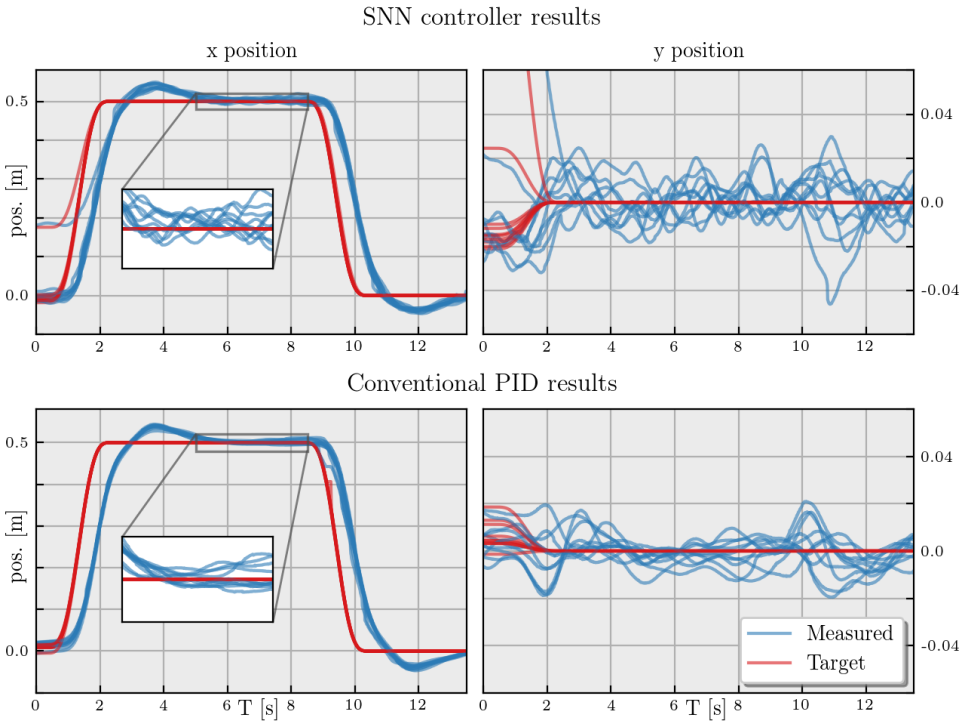


Figure 5.8: Position setpoint responses of both the SNN controller (top) and the conventional PID (bottom). For both, the positions as measured by the Motion Capture system are shown in blue, and setpoints in red. Although the SNN controller has a slightly noisier response, its trajectory is very similar to the PID.

make the spiking representation of the input signals more accurate.

Besides the position-control tests, the response to manual flight was performed with which the response of the different parts of the controller on a real-world system could be analyzed. In Figure 5.9, one second of such a test is shown. It is evident that the proportional part is very accurate, and the integral pathway is able to effectively deal with prolonged errors. The response of the derivative pathway is less accurate. This is most likely caused by the mechanisms shown in the derivative analysis, earlier in this work (Section 5.3.1), where it was shown that the derivative pathway tunes to specific delays. Even though the network is trained on real flight data, the range in delays might be too large which results in larger errors for the lowest and highest frequencies. For control of the Crazyflie, this is still acceptable since the derivative control path still effectively dampens the response by countering large derivatives in the input.

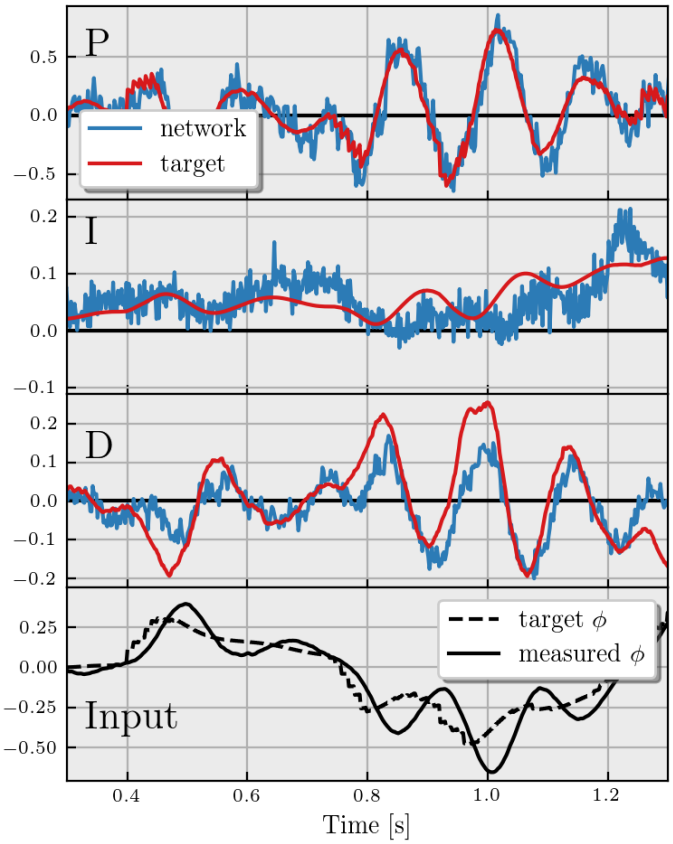


Figure 5.9: Output of the SNN controller in blue versus target values computed with a conventional PID controller in red for a manually controlled flight. The three uppermost figures depict the individual components of the controller. As can be seen, the SNN controller produces very similar results to the reference controller. The lowermost figure shows the input to the network; the target rotational rate ϕ and the rate measured by the gyro.

5.5. Conclusion

In this work, we have proposed a novel input threshold adaptation mechanism, Input-Weighted Threshold Adaptation (IWTA). This mechanism adds extra weights per input connection that regulate the spiking threshold of the LIF neuron. By doing so, it enhances the network with the ability to integrate information over time, something the regular LIF model is unable to do.

Also, we have shown that neuromorphic controllers using rate-based encoding can be used to control highly unstable underactuated systems. To demonstrate this, we have shown control of the innermost loop of a real flying tiny quadrotor, the Crazyflie. Using only 320 neurons per control axis, the network showed to be capable of stable and robust control, with the potential of extremely low delays due to the high inference speed of neuromorphic hardware. By a straightforward training method using surrogate gradients and Backpropagation Through-Time, the network can be fine-tuned to a very limited amount of data from a real-world flying drone. Due to the sparse connections, the network is able to optimally benefit from the advantages of neuromorphic hardware.

In future work, we intend to apply the IWTA mechanism to different tasks and benchmarks to further establish its potential. Even though the different time constants for the derivative neurons allow us to dampen the control response, we have shown that they are limited to a specific frequency. We will also investigate the application of IWTA on the derivative neurons to improve the accuracy over a much broader range of frequencies. The availability of a neuromorphic controller, such as a PID, that can be easily implemented on neuromorphic hardware plays a crucial role in completing the neuromorphic control loop in robotics. These controllers can be readily integrated into pipelines utilizing event-based algorithms, like a vision-based control system using an event camera as in [59]. Lastly, the fast-paced and unpredictable movements of drones demand high-performance computing that traditional hardware struggles to provide, making neuromorphic processing an attractive alternative.

Supplementary information

5.5.1. Current-based Leaky-Integrate-and-Fire neuron

The current-based Leaky-Integrate-and-Fire (CUBA-LIF) is widely used in literature, available in most SNN simulators and commonly used in neuromorphic hardware, such as Intel's Loihi [19]. The discrete-time dynamic equations of the LIF neuron are as follows:

$$v_i(t+1) = \tau_i^{\text{mem}} v_i(t) + i_i(t), \tag{5.9}$$

$$i_i(t+1) = \tau_i^{\text{syn}} i_i(t) + \sum w_{ij} s_j(t), \tag{5.10}$$

where $v_i(t)$ is the membrane potential at time t, $\tau_i^{\text{mem}} \in [0,1]$ and $\tau_i^{\text{syn}} \in [0,1]$ the membrane and synaptic time constants, $i(t)$ the synaptic current at time t, w_{ij} the synaptic weight between neurons i and j, and s_j a binary value representing either

a spike or no spike coming from the pre-synaptic neuron j. To determine whether a neuron emits a spike, the membrane potential is reduced with the neurons firing threshold θ_i^{thr} and passed through the Heaviside step function to determine the output of the neuron:

$$s_i(t) = H(v_i(t) - \theta_i^{\text{thr}}) = \begin{cases} 0, & v_i(t) - \theta_i^{\text{thr}} \leq 0 \\ 1, & v_i(t) - \theta_i^{\text{thr}} > 0 \end{cases} \tag{5.11}$$

When the Heaviside function resolves to 1 and the neuron emits a spike, the membrane potential $v_i(t)$ is reduced by the threshold value (in literature this is known as a soft reset [101]).

5.5.2. Trained parameters and ranges

In Table 5.1, all the parameters used in the network are given, along with the specified range and the number that was used in the Crazyflie application.

	Parameter	Range	Count
P	τ_i (current decay)	[0, 1]	80
	τ_v (voltage decay)	[0, 1]	80
	w_i (input weight)	[0, ∞]	40
	w_o (output weight)	[0, ∞]	40
I	τ_i (current decay)	[0, 1]	80
	τ_v (voltage decay)	[0, 1]	80
	w_i (input weight)	[0, ∞]	40
	w_o (output weight)	[0, ∞]	40
D	τ_i (current decay)	[0, 1]	160
	τ_v (voltage decay)	[0, 1]	160
	w_i (input weight)	[0, ∞]	80
	w_o (output weight)	[0, ∞]	80

Table 5.1: All parameters trained in the network, their given range, and how many are used for the real-world tests.

Link to paper:

6

Neuromorphic attitude estimation and control

To reap the maximal benefits from neuromorphic computing, it is necessary to perform all autonomy functions end-to-end on a single neuromorphic chip, from low-level attitude control to high-level navigation. This chapter presents the first neuromorphic control system using a spiking neural network (SNN) to effectively map a drone's raw sensory input directly to motor commands. We apply this method to low-level attitude estimation and control for a quadrotor, deploying the SNN on a tiny Crazyflie. We propose a modular SNN, separately training and then merging estimation and control sub-networks. The SNN is trained with imitation learning, using a flight dataset of sensory-motor pairs. Post-training, the network is deployed on the Crazyflie, issuing control commands from sensor inputs at 500Hz. Furthermore, for the training procedure we augmented training data by flying a controller with additional excitation and time-shifting the target data to enhance the predictive capabilities of the SNN. On the real drone, the perception-to-control SNN tracks attitude commands with an average error of 3.0 degrees, compared to 2.7 degrees for the regular flight stack. We also show the benefits of the proposed learning modifications for reducing the average tracking error and reducing oscillations.

Parts of this chapter have been published as an IEEE Robotics and Automation Letter (RA-L) 2025 [60]

6.1. Introduction

Quadrotors have soared in popularity over the past decade, significantly influencing the field of unmanned aerial vehicles (UAVs) with their unique capabilities. These agile machines are applicable in a myriad of applications, such as search and rescue operations [102], environmental monitoring [103] and precision agriculture [104], owing to their ability to hover, perform vertical take-offs and landings, and navigate through confined spaces with remarkable precision.

The integration of Artificial Intelligence (AI) promises to extend the capabilities of quadrotors even further [105, 106]. By leveraging advances in AI, we can envision quadrotors that not only perform pre-programmed tasks but also adapt to new challenges, achieving levels of flight performance and operational robustness previously unattainable while solving tasks that are currently performed post-flight or offboard. However, the current generation of quadrotors is hindered by hardware that is often power-hungry and algorithms that fall short in efficiency and adaptability [107].

A promising solution to these challenges lies in the emerging field of neuromorphic hardware [108]. Neuromorphic systems, including processors and sensors such as event-based cameras [20, 109], draw inspiration from neural systems found in nature. These systems use sparse and asynchronous spikes to transmit information that are both energy-efficient and enable high-speed processing. Due to the low latency, this approach is particularly well-suited for dynamic environments where rapid decision-making is crucial [110]. Central to this neuromorphic paradigm are Spiking Neural Networks (SNNs) [111], which emulate the brain's information processing using neural spikes. SNNs have demonstrated their potential in various robotic applications, yet their use in controlling the full flight dynamics of quadrotors remains largely unexplored. By adopting strategies seen in nature, such as the reflexive control and visual processing used by the fruit fly [89], we can develop more integrated and efficient control systems. This does, however, require a fully end-to-end neuromorphic system.

Neuromorphic control is a nascent field at the intersection of neuroscience and robotics control theory [112]. The benefits of neuromorphic hardware, such as fast inference and high energy efficiency [86], harmonize with demanding control and estimation tasks. While the output of rudimentary sensors for quadrotors, such as Inertial-Measurement-Units, can already be processed at the high frequencies necessary for agile and robust control, vision-based tasks are

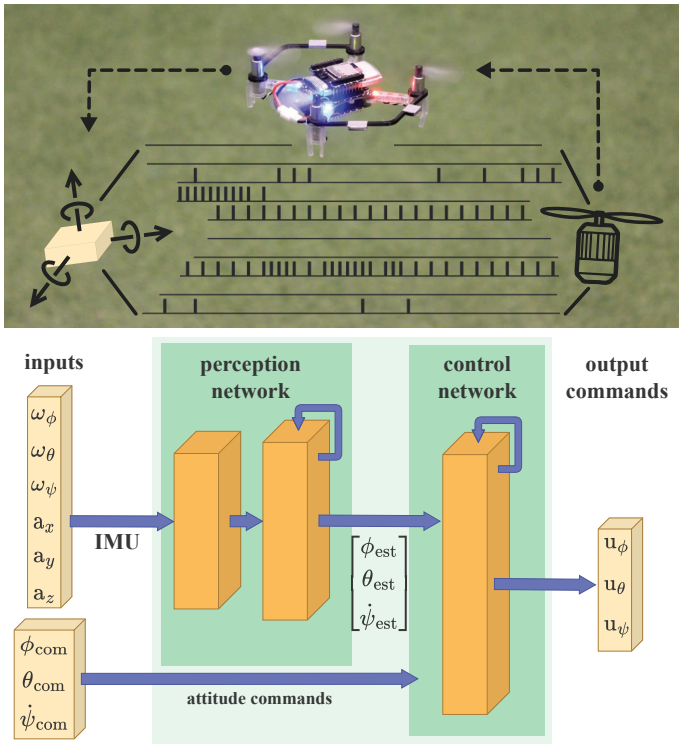


Figure 6.1: We present an approach to training a spiking neural network for end-to-end attitude estimation and control of tiny drones (deployed on a Crazyflie, top). The network is a merging of a 2-layer attitude estimation sub-network with recurrency and a 1-layer recurrent attitude control network (bottom). The network exhibits a spiking activity of 15%, which is promising in terms of energy efficiency for future implementation on a neuromorphic processor. The network currently runs at 500Hz on a Teensy microcontroller.

severely limited by processing power on a flying machine [113]. However, Dimitrova *et al.* [114] have shown that using event-based cameras allows a quadrotor to track the horizon at extremely high speeds. To further increase the potential of such a system, the authors of [22] showed that integration of this horizon tracker with a manually-tuned SNN controller on a single neuromorphic processor leads to even faster control, benefiting from having all parts on the same chip.

Despite significant advances in AI for quadrotors, limitations remain, particularly in vision-based tasks constrained by onboard computational resources. Falanga *et al.* [115] argue that regular frame-based cameras are

inadequate for avoiding obstacles due to their high latency, which can be detrimental in fast-paced environments. Although event-based cameras address these latency issues, the processing on non-neuromorphic hardware required compromises in detection algorithms to favor speed over accuracy.

Recent breakthroughs in quadrotor research have achieved impressive results, such as outperforming human pilots in drone races using only onboard computations [116]. Also, Song *et al.* [105] show that for these tasks, optimal control methods are no longer sufficient and are beaten by Reinforcement Learning (RL) employing Deep Learning.

Despite these accomplishments, the reliance on slower frame-based vision systems, typically operating at 30Hz or lower, highlights a significant gap where neuromorphic solutions could offer substantial improvements. These examples underscore the critical need for fully integrated neuromorphic systems capable of high-speed data processing.

To allow such a unified system, the entire estimation and control loop needs to be considered. Despite the promising results in partial implementations, a fully integrated neuromorphic system connecting sensor inputs directly to motor commands has not yet been realized in operational quadrotors. Results focusing exclusively on lower-level SNN control have been obtained using manually tuned networks [26, 66] or were limited to simulation [90, 91]. Moreover, even state-of-the-art learned quadrotor controllers using regular Multilayer Perceptrons (MLPs) as presented in [9, 117] and [118], that were learned with RL, assume full state knowledge or need a lower-level controller to go from rate commands to motor outputs. Zhang *et al.* [119] have demonstrated in simulation that by using an expert privileged policy, an MLP can be trained to perform end-to-end control. But also here the observation model, containing the measurements, included a direct measurement of the drone's attitude. However, such privileged information – complete and accurate state information – is rarely available in real-world scenarios. This limitation is further exacerbated by the reality gap, that arises when algorithms trained or evaluated in simulation must cope with real-world conditions characterized by imperfect measurements, sensor noise, actuator delays, and unpredictable environmental influences.

Notable efforts towards a complete end-to-end neuromorphic system include the use of Intel's Loihi processor [19] in a quadrotor for velocity control based on optical flow estimates from event-based cameras [59], which successfully combined ego-motion estimation with a basic linear controller. The experimental results confirmed the potential of neuromorphic technology, as the vision ran at frequencies between 274-1600Hz, while only spending 7mW for network

inference compared to 14-25Hz on a Jetson Nano that required 1-2W for inference. The neuromorphic system was not only significantly faster, but also required orders of magnitude less power. However, it still relied on a companion computer for attitude control, introducing delays, increasing power consumption, and adding weight to the drone. Moreover, the linear neuromorphic controller lacked a mechanism to compensate for steady-state errors, such as those caused by sensor biases like gyroscope drift. With our work, we want to demonstrate how the pipeline of [59] could be extended to run on a single neuromorphic chip. In [120] a closed-form spiking network was proposed that could do end-to-end control and estimation for linear systems and was shown to perform well with a small number of neurons in simulation. Since this approach needs to be able to read out a floating point "firing rate" of neurons in the hidden layer, it is not trivially implemented on commonly available neuromorphic hardware where the input and outputs are limited to vectors of binary spikes.

The main contribution of this article is that we design, train, and implement the first fully neuromorphic system for attitude estimation and control of quadrotors. The proposed method involves real-time processing from sensors to actuators and does not require traditional computing hardware. Our approach is to train two separate sub-networks, one for state estimation and one for control, and to merge them after training. For both parts of the network, we employ supervised / imitation learning. In our creation of the training scheme we had to overcome substantial challenges, as the spiking neural network needs to cope with (i) sensor bias, (ii) delays due to the progressive updates of spiking neural networks, (iii) the reality gap and (iv) converting binary spikes to a motor command that leads to smooth control. Additional contributions of our work concern how we tackled these challenges. For the sensor bias, we find that constraining the parameters of a small subgroup of neurons to function as integrators is necessary for successful training results. These integrator neurons can now operate analogously to the integral component of a standard PID controller, effectively mitigating persistent sensor biases. For the delays in the SNN, we propose to time-shift the targets for learning, so that the SNN predicts future outputs of the traditional controller. This brings substantial performance improvement. For the reality gap, we first add noise to the motor outputs of the traditional controller to sufficiently excite the system and avoid biases in learning. Subsequently, we gather more training data with a first version of the SNN, so that relevant off-target attitudes and rates are explored. Finally, we evaluate system performance in real-world conditions, comparing the trained SNN with traditional control methods.

The remainder of the article is structured as follows. Section 6.2 details our

methodology, covering attitude control from sensor data, the network architecture, training procedures, and the hardware used for real-world testing. In Section 6.3, we present the test results, including position control, attitude control, and an analysis of power consumption. Finally, Section 6.4 summarizes our key findings and outlines potential directions for future work in neuromorphic control systems.

6.2. Methodology

This section discusses how an SNN used for attitude estimation and control of the Crazyflie in real time, was constructed and trained.

6.2.1. Attitude control from IMU measurements

The attitude of a quadrotor, its orientation relative to gravity, can be estimated using measurements from an Inertial Measurement Unit (IMU). These IMUs commonly contain a 3 DOF (Degree of Freedom) gyroscope, measuring rotational velocities and a 3 DOF accelerometer, measuring linear acceleration. The gyroscope data offers high-frequency information about the rotation of the quadrotor while the accelerometer measurements contain an absolute measurement of the gravity vector [121]. Combined, these two form the backbone of most quadrotor control algorithms. These 6 inputs are usually combined into an estimate of the orientation of the drone, which in turn gets sent to a controller together with a target orientation. This controller calculates the necessary motor speeds for each four rotors.

6.2.2. Spiking neural network architecture

LIF neurons

In this work, we apply one of the most common spiking neuron models; the current-based leaky-integrate-and-fire (CUBA-LIF) neuron. This model is chosen since it captures temporal dynamics, is computationally efficient and is the default model in current available neuromorphic platforms such as Intel's Loihi [19]. Each neuron is connected to other neurons via *synapses*, connections that carry a multiplicative weight. Every neuron keeps track of two hidden states at each timestep; its *membrane potential* and *synaptic current*. The membrane potential v and synaptic input current i at timestep t as discrete functions of time

are given as:

$$v_i(t+1) = \tau_i^{\text{mem}} v_i(t) + i_i(t), \tag{6.1}$$

$$i_i(t+1) = \tau_i^{\text{syn}} i_i(t) + \sum w_{ij} s_j(t) + \sum w_{ik} s_k(t), \tag{6.2}$$

where j and i denote presynaptic (input) and postsynaptic (output) neurons within a layer, k the neurons in the same layer as i, $s \in [0, 1]$ a neuron spike, and w^{ij} and w^{ik} feedforward and recurrent connections (if any), respectively. The leak values of the two internal state variables are denoted by τ_i^{mem} and τ_i^{syn}. A neuron fires an output spike if the membrane potential v_i exceeds threshold θ_i to all connected neurons, resetting its membrane potential to zero at the same time.

The input of the networks during training is a linear layer that is directly inserted into the current i of the first layer. This way, the encoding of floating point sensor data to binary spikes is included in the training procedure. The output is decoded similarly; the hidden spiking layer is connected via a weight matrix to the outputs.

Combination of networks

To facilitate learning of specific tasks and increase the debugability, the training is split into two parts; estimation and control. By learning layers of spiking neurons that have a certain function, there is more control over the stability of the final solution, and it also reduces the search space. Since we define the input- and output values of both sub-networks as a linear multiplication of the input- or output-vector respectively, the networks can be easily combined. The output of the first network can be written as $y(t) = W_{\text{o}} s(t)$, with $s(t)$ the spikes in the hidden layer, and the input to the next network is $x(t) = W_{\text{i}} y(t)$. We can now combine these by multiplying the weight matrices of the output weights W_{o} of the first network and the input weights W_{i} of the second, as introduced in [59], since these are both linear transformations. The attitude part of the input to the second network can therefore be written as

$$\begin{bmatrix} \phi_{\text{est}} \\ \theta_{\text{est}} \\ \psi_{\text{est}} \end{bmatrix} = W_{\text{i}} W_{\text{o}} s(t). \tag{6.3}$$

Stacking the binary output spikes of the first network with the floating-point command values that are passed (see Figure 6.1), the new set of weights to the hidden layer of the second network can be written as

$$W_{\text{new}} = \begin{bmatrix} 0 & W_{\text{i, command}} \\ W_{\text{i}} W_{\text{o}} & 0 \end{bmatrix}. \tag{6.4}$$

6.2.3. Training

The model is trained using imitation learning, cloning the behavior of an expert policy. Data is gathered at 500Hz by flying manually with a Crazyflie for 20 minutes. During these tests, the Crazyflie uses a complementary filter for estimating the attitude and a cascaded PID controller for control. In this work, these function as the expert policy. The Crazyflie controller used the default parameters as defined by the Bitcraze firmware [100]. This data was split into sequences of 2000 timesteps and normalized according to total training set statistics. From every sequence the integrator value at the beginning of this sequence was subtracted, since this value is not contained in the input data so would not be possible to learn. All of the parameters p of the network (τ_i^{mem}, τ_i^{syn}, w_{ij}, w_{ik} and θ_i) were then trained using supervised backpropagation-through-time (BPTT). The loss was defined as a weighted sum of the Mean Squared Error (MSE) and the Pearson Correlation Loss;

$$J(p) = \text{MSE}(x, \hat{x}) + \frac{1}{2}(1 - \rho(x, \hat{x})), \tag{6.5}$$

with x and $\hat{x}$ the target- and network response values respectively and $\rho(x, \hat{x})$ the Pearson Coefficient [99]. One major step in training SNNs using regular BPTT despite the non-differentiability of the spiking threshold function is replacing the Heaviside step-function in the backwards pass with a surrogate function that represents a smooth approximation of the real gradient [97]. In this work, the derivative of a scaled arctangent was used, like in [98];

$$\frac{d}{dx}\left(\frac{1}{s}\arctan(sx)\right) = \frac{1}{1 + (sx)^2}, \tag{6.6}$$

where s is the slope of the surrogate. A higher slope results in a more accurate proxy of the real gradient, but can lead to vanishing gradients for neurons with a very low or high membrane potential. A shallow slope, on the other hand, is less accurate but leads to less ”dead” neurons that have no contribution to the output. Among alternatives for the surrogate gradient is the derivative of the Sigmoid, but research has shown that the exact shape does not matter [33]. The slope s of the derivative, however, does have a large influence on the training speed and final results. For this work, the slope s has been set to 7.

Multiple challenges were observed during the training/deployment iterations. These are discussed here.

Delay in SNN, training with time-shifted data

During training-implementation iterations, oscillations were observed on the real quadrotor. After investigation, these were attributed to a delay in the output of the network versus the target control signal. Due to the nature of the SNN with the implicit memory due to the leaking voltage and current, the output was delayed. This can be observed in Figure 6.2. In the top part of the figure, the Pearson Correlation between the output of the SNN and the regular PID is compared for different shifts in time on the entire data set. In the bottom part of the figure, a small time sequence is shown that clearly shows the lag. The correlation is highest for 5-6 timesteps shift, indicating that this is indeed a problem when one trains SNNs for highly dynamic tasks that require a quick response to fast changes. In the case of a controller, a small delay in the derivative command will induce oscillations. To reduce this delay, and improve

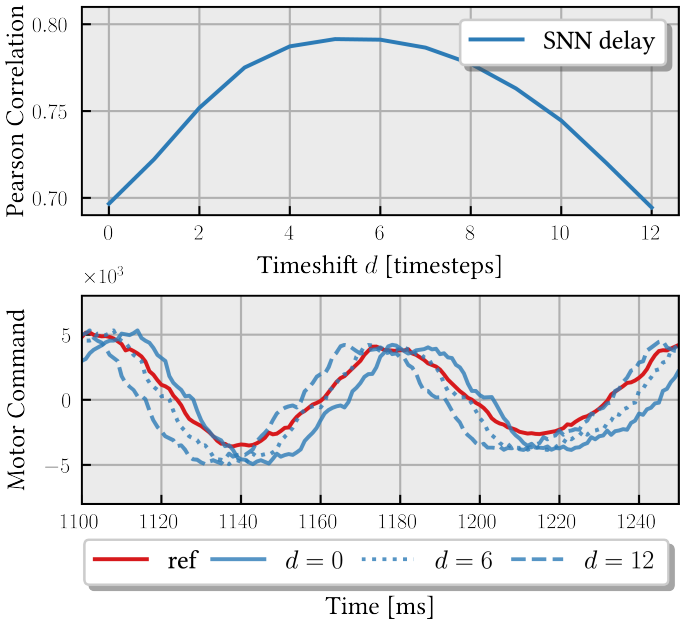


Figure 6.2: Pearson Correlation between the output of the trained SNN and the regular PID output for different time shifts d. The bottom graph shows the output of the network for time shifts $d = 0$, $d = 6$ and $d = 12$ compared to the target, further demonstrating that a delay is present in the network.

flight characteristics, we trained the control network on a time-shifted version of the target data. Specifically, we used the target signals of ≈ 6 steps in the future. Consequently, the SNN needs to predict the reference control output in the

future, which in turn results to less delay in the implemented controller.

Imitation learning; reducing the reality gap

The reality gap is a significant challenge in imitation learning particularly, since the reference controller only explores a limited portion of the state space around its stable behavior. This leads to a dataset that does not fully represent the full range of potential flight conditions or disturbances the SNN controller may encounter when deployed [122, 123]. Consequently, when the trained controller operates in real-world conditions, it can encounter ”unseen” states or disturbances not present in the training data, resulting in unpredictable and unstable behavior.

To address this, we expanded the training data to include a broader, more realistic range of states. Initially, the SNN controller was trained on data generated with the reference controller in the loop, as described in Section 6.2.3. We then conducted additional data collection in two steps to diversify the training set: (1) flying the quadrotor with the initially trained SNN controller in the loop, while simultaneously logging the outputs the reference controller would have provided. This approach exposed the SNN to a set of states it is likely to encounter, fine-tuning the network around these points. (2) Introducing random disturbances to the regular PID controller’s outputs to simulate unexpected environmental or system changes. Specifically, disturbances were applied to pitch, roll, and yaw commands at a 1% probability per timestep (at 500Hz), lasting 0.2 seconds each, with disturbance size $X \sim U(0, 50)\%$ of the absolute maximum command.

This additional data, including both the reference controller outputs and the effects of random disturbances, was incorporated into the training set. Retraining the SNN controller on this expanded dataset improved its robustness, enabling it to generalize across a wider range of states and disturbances, thereby reducing the likelihood of instability during real-world deployment.

Splitting estimation and control

As discussed in the section on architecture (see Figure 6.1), the network was split into an estimation and control part. If the network learning attitude estimation also has access to the control command, training is prone to end up at a local minimum. The network will then learn a function between control command and attitude; since the reference controller was in the loop this will be an easy function to learn. It can then completely disregard the sensor data, or only use it to slightly

optimize the estimation. When this estimator is then used in the loop, the function between input command and attitude will be different since the trained controller is not perfect; this will further degrade flight performance. Hence, no connections between the input command and the attitude estimation layer are established.

Integrator

In developing an integrator within the spiking neural network (SNN) architecture, we faced challenges with parameter sensitivity, where small adjustments often led to significant errors or instability, causing the network to either underestimate the integral or diverge. This challenge is particularly acute in recurrent neural networks (RNNs), where recurrent gains above 1 often destabilize the system, while a recurrency lower than 1 produces a low-pass filter response. Orvieto *et al.* [124] have shown that carefully structuring RNN network architecture before training (e.g. by linearizing and diagonalizing the recurrency) is important to obtain the superior results of deep State Space Models (SSMs) [125].

Another issue was the integrator signal's dynamic: it shows large deviations at the start of a flight test but stabilizes quickly under constant disturbance. Effective integration through imitation learning required varying disturbances and resetting the initial integral for each sequence. Additionally, the integral signal changes more slowly than the proportional and derivative components, complicating the extraction of integral information from the total signal in a supervised-learning scheme.

To address these issues for SNNs, we propose fixing certain neuron parameters within a small subgroup of neurons during training to ensure stability. Specifically, we set the leak parameters τ_i^{syn} and τ_i^{mem} and threshold θ_i of 10 neurons in the control layer to 1. This allowed the neurons to integrate incoming signals without decay. By training only the input and output weights and averaging spike outputs on integral data alone, we achieved a spike rate approximating the cumulative incoming signal, making the neuron responsive to transient and steady-state inputs. This approach is validated in Figure 6.3, which compares training curves for an integration task with fixed versus free neuron leak and threshold parameters. The fixed-parameter integrator provided the necessary stability, outperforming the fully unconstrained trained approach and satisfying the SNN-based system's control requirements.

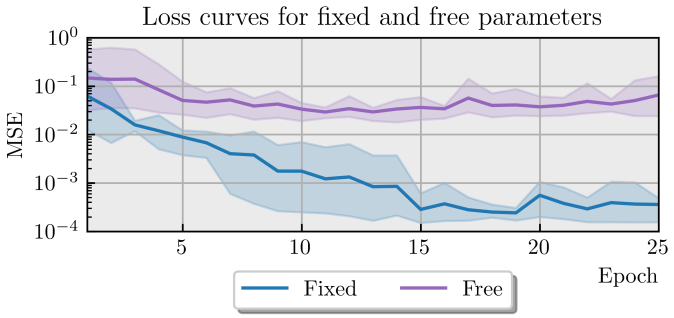


Figure 6.3: Training loss curves comparing fixed versus free neuron leak and threshold parameters. The proposed approach of fixing neuron parameters leads to stable convergence during training. Allowing these parameters to remain free results in training becoming trapped in local minima.

6.2.4. Hardware setup

To demonstrate the capabilities of our approach, we have implemented it in the control loop of the tiny open-source quadrotor Crazyflie [100]. By adding a Teensy 4.0 development board to the Crazyflie, the necessary computation power for running an SNN on a processor was obtained. This allowed us to run the complete SNN from input encoding to control commands at 500Hz in C++ on the ARM Cortex-M7 microprocessor. To carry the extra weight of the Teensy, the regular 16mm brushed motors of the Crazyflie are swapped with 20mm brushed motors. To maximize the accuracy of the network while utilizing the Teensy to its full extent, the network was optimized for speed by removing unnecessary neurons. This was done by performing inference on a number of test sequences and calculating the total contribution of a neuron on the output by calculating the total number of spikes emitted multiplied by its weight to all outputs. Now the N lowest contributing neurons can be removed from the implementation in C++ on the Teensy. Although the network was trained with 150-150-130 neurons per layer respectively, we reduced the size to 150, 100, and 80 per layer respectively. By mainly pruning the neurons with recurrent connections this way, we almost halve the number of mathematical operations while retaining over 99% of the original MSE that was used as the loss function during training.

We send the attitude setpoints, along with the IMU measurements from the gyroscope and accelerometer, via UART to the Teensy deck. The neural controller's torque command outputs are transmitted back to the Crazyflie through the same UART connection, where they are incorporated into the motor mixer. The motor mixer is a linear transformation that converts torque

commands into rotor velocities. As the network runs at 500Hz in the loop, the maximum delay introduced in the system is 2 milliseconds. Even though this is fast enough to keep up with the lower-level control-loop in the Crazyflie, it might still influence the overall stability.

An OptiTrack motion capture system provides accurate position measurement and an absolute heading. These are sent to the Crazyflie via a radio connection to a ground station laptop, which also handles the sending of high-level commands.

The total take-off weight of the Crazyflie, including the Teensy 4.0 and upgraded motors, is only 35 grams. This allows for approximately 5 minutes of flight time.

6.3. Results

6.3.1. Position control

To demonstrate the capabilities of the proposed SNN, we include it in a position control task. The higher-level attitude commands together with the IMU values are sent as inputs to the SNN, which produces pitch, roll and yaw torque commands. After a short period of hovering at $(x, y) = (0, 0)$, the Crazyflie is commanded to move 1 meter in x-direction after which it is commanded to move back to $(0, 0)$. For both the SNN and PID controller, these tests were performed ten times. In Figure 6.4, the position control results are shown. The results show that performing attitude estimation and control using an onboard SNN results in stable reference tracking, comparable to the regular flight stack of the Crazyflie.

6.3.2. Impact of time-shifted and augmented training data on SNN performance

During testing, it was quickly identified that training the fusion network without augmenting the dataset does not produce a network that can be used in flight. Therefore, it was necessary to augment the dataset for this sub-network. However, to further investigate the behavior of the SNN and the influence of the modifications to the training procedure, another test is performed. Since the directly controlled variable is the attitude command, we compare the response of differently trained networks to an attitude setpoint change. For these tests, the Crazyflie received a roll setpoint of 0° for 2 seconds, followed by a setpoint of $+10^\circ$ for 1.5 seconds, a setpoint of -10° for 1.5 seconds before returning to a 0° setpoint for 2.5 seconds. Again, we performed ten tests per controller. The combined results of these ten tests per controller are shown in Figure 6.5, with A) the final SNN, B) the SNN that was trained on the augmented dataset, C) the SNN

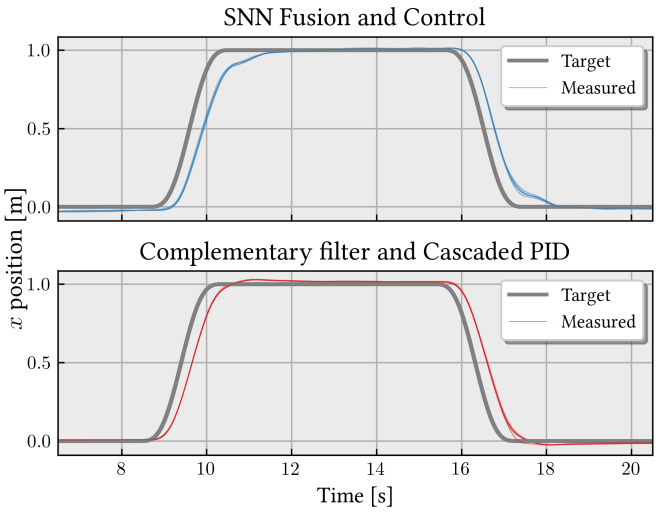


Figure 6.4: Position step responses of the SNN system (top) and the regular PID flight stack (bottom) for 10 individual test runs. The SNN can accurately track the attitude references as given by the outer-loop position controller and maintain a stable flight path.

that was trained on time-shifted data, but without augmenting the dataset and D) the regular attitude estimator and controller on the Crazyflie. The Root Mean Square Error (RMSE) between the commanded roll setpoint and the resulting (estimated) roll angle is given in Table 6.1, together with the average standard deviation (SD) of the response with respect to the average of all tests with the same controller. With a tracking error of only $3.03°$, the network is able to correctly estimate the attitude and also control it. Adding the suggested modifications to the training procedure reduces the tracking error from $3.24°$ to $3.03°$ compared to $2.67°$ for the reference controller (please note that the reference controller receives the estimated attitude directly, while the SNN needs to internally calculate this). Also, training on time-shifted data significantly reduces the oscillations as can be seen in Figure 6.5. This can also be inferred from the average SD that is significantly lower for the fully-trained SNN, showing that the controller performs more consistently across multiple tests. On the other hand, training on time-shifted data very slightly increases the rise-time (see Table 6.1). Since the increase is in the order of milli-seconds, it will not affect tasks like obstacle avoidance that generally operate in the 20-40Hz range [126] but it should be considered if it is used in super agile flight.

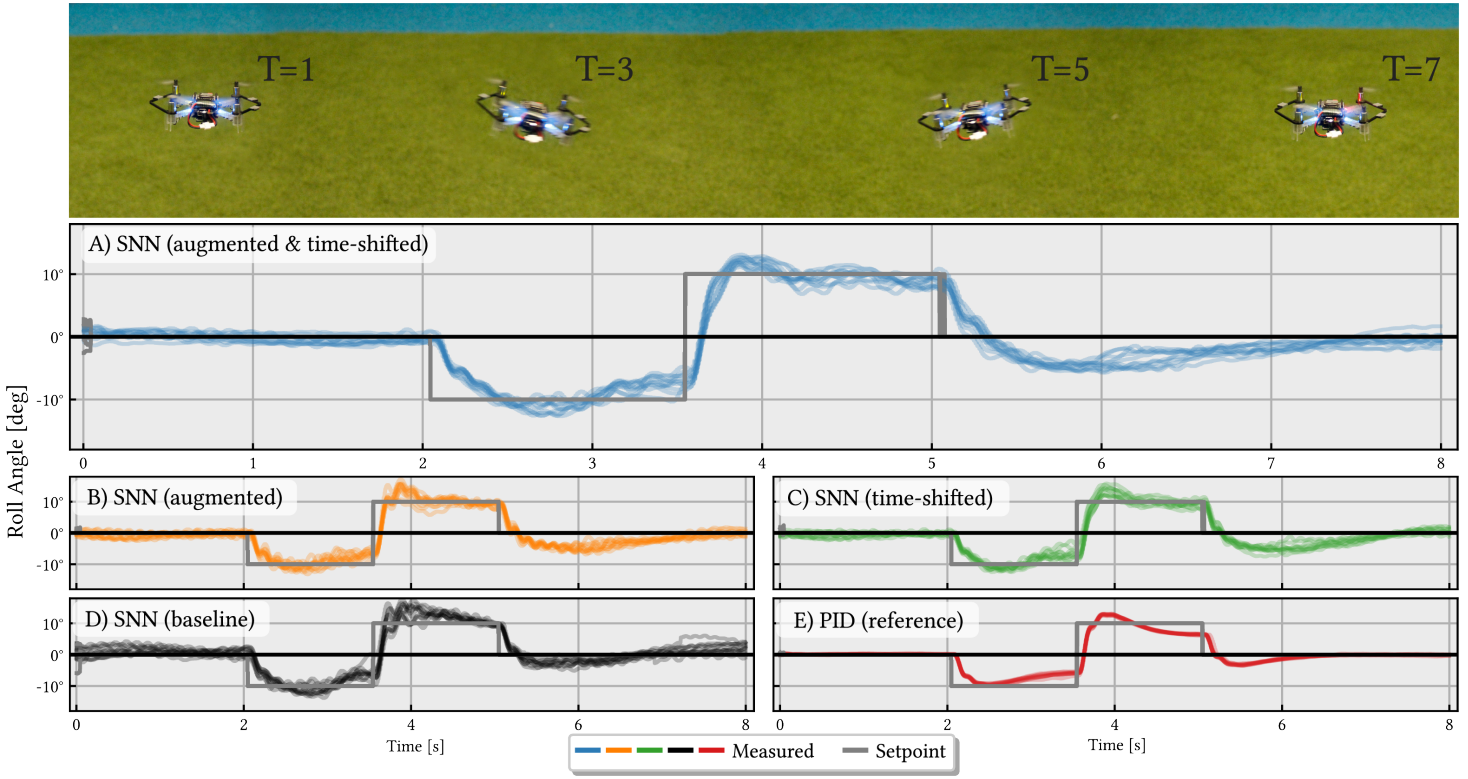


Figure 6.5: Attitude step responses of A) the fully-trained SNN system, B) the SNN trained with augmentation, C) the SNN trained with time-shifted data and D) the regular PID flight stack. The images on top show the Crazyflie during the different maneuvers.

Controller	RMSE	avg. SD	avg. RT
SNN (time-shifted & augm.)	3.03°	0.77	145ms
SNN (augmented)	3.10°	0.95	130ms
SNN (time-shifted)	3.24°	0.92	145ms
SNN (baseline)	3.14°	1.16	135ms
PID	2.67°	0.23	125ms

Table 6.1: Root Mean Square Error (RMSE), Standard Deviation (SD) and rise-time (RT) comparison between different controllers. Note that the PID receives the estimated attitude as input, while the SNN needs to calculate this internally.

6.3.3. Power usage analysis

The main benefits of having an end-to-end attitude SNN mainly derive from its combination with other autonomy functions such as computer vision on a single neuromorphic chip. Given the elementary nature of attitude estimation and control tasks, we do not expect any substantial performance or energy improvements for attitude estimation and control by itself.

Still, we do think it is insightful to analyze the power usage of the current solution. The SNN in this research runs on a conventional microprocessor, as currently available neuromorphic chips (like Intel's Loihi [19, 127] or SpiNNaker [17]) require supporting embedded systems that are too large for a 35-gram quadrotor or challenging to source. To explore potential power advantages, we performed some estimative calculations. Spike propagation through the network relies solely on additions rather than multiplications, allowing us to calculate the necessary operations based on addition alone. For the three-layer network used here, this would initially amount to approximately 42,500 additions per update. However, due to the 15-20% sparsity in neuron activations at each timestep, the actual required operations reduce to around 7,500 additions. In contrast, the cascaded PID controller on the Crazyflie requires about 28 additions and 52 multiplications per timestep. Moreover, a straightforward complementary attitude estimation filter will have as most expensive operation a non-linear `atan2` function that requires in the order of 15-30 multiplications. Since a 32-bit floating-point multiplication uses roughly 37 times more energy than a 32-bit integer addition [128], we can roughly equate the number of additions of a straightforward traditional pipeline with $\approx$ 3,000 additions. Hence, on a conventional microcontroller, the SNN performs in the same energy order of magnitude as a PID-based controller.

If small neuromorphic hardware becomes available that can natively support IMU readings, while implementing the SNN in hardware, energy consumption can be substantially reduced. Nonetheless, we maintain that the real gain would come when expanding this network to handle image data for instance, as seen in other neuromorphic works that show up to 100 $\times$ gains in efficiency (e.g. [59, 127]). This would create larger disparities due to the high multiplication demands in image processing tasks. Then, implementing all functionality in a single neuromorphic chip would make conventional companion computers obsolete, massively reducing energy consumption.

Finally, further benefits can be expected when moving to event-based control, which has demonstrated potential for drastic reductions in computational load (up to 80% for quadrotor attitude control [129]) by activating only when significant events occur. A drone in hover should only need to interfere and adapt its actuator commands when it starts to move, requiring no energy expenditure in between control events. Current microprocessors can not optimally benefit, because they still need to perform operations at a fixed frequency.

6.4. Conclusion

In this article, we have presented the first fully spiking attitude estimation and control pipeline for a quadrotor. We show that by using imitation learning, it is possible to train a fully end-to-end SNN to control a micro drone. We augmented training data to further enhance the performance, using in-flight data. The network was also taught to predict a k-step advance control action to mitigate delays that are inherent to the SNN. These methods led to significant reductions in RMSE relative to the target attitude and decreased oscillations, collectively enhancing the drone's flight stability. Furthermore, our findings indicate that constraining parameters during training to function as integrators improves training precision and information integration. For RNNs these parameters would be the recurrent weights, and for SNNs the leak and threshold parameters. This novel approach avoids local minima during training and allows for faster convergence. Next to that, our methods of implicitely learning integration and differentiation are not only applicable to attitude control for quadrotors, but apply to perception and control for robotics in general (e.g. using integration with rotary encoders or using differentiation to predict future states in model-based control). By evaluating the system's performance in real-world conditions and comparing it with traditional control methods, we have laid the groundwork for future developments in neuromorphic control strategies. The importance of a

working imitation learning pipeline, for instance, has been demonstrated in [57], where the authors show that bootstrapping a RL pipeline with imitation learning results in more reliable RL training while outperforming imitation learning only. Our methods can thus be used to improve RL for SNNs.

Future research should aim to implement these algorithms on neuromorphic hardware, which could yield substantial gains in energy efficiency and reduced latency, potentially extending flight times and enabling neuromorphic UAVs in energy-constrained scenarios. By advancing these techniques, we envision the next generation of highly efficient, adaptive, and intelligent UAVs.

Supplementary materials

All code necessary to 1) train the SNN, 2) convert and run the SNN on a Teensy 4.0, 3) integrate in the Crazyflie firmware and 4) perform the tests can be found in:
`https://github.com/tudelft/neuromorphic_att_est_and_control`.
The data that was used for training can be found in:
`https://doi.org/10.4121/f474ef0a-6ef1-4ea1-a958-4827c4eadf60`.

Link to paper:

7

Conclusion

This thesis set out with the goal to investigate - and deal with - the issues that arise with the implementation of SNNs and other neuromorphic algorithms in the lower level control loop of flying vehicles. In this concluding chapter, we will first look backwards before looking at the future of neuromorphics in the field of flying robots.

7.1. Research goal and questions

We started by looking at *state estimation*, as was posed in Research Question 1 as follows:

> **Research Question 1**
>
> How can we develop and train a neuromorphic system to estimate the attitude of a flying drone?

This was answered by using recurrent networks of spiking neurons with IMU-sensor data as input in Chapter 2 and by combining the outputs of a downward facing event-camera in Chapter 3. In Chapter 2, we showed that a recurrent SNN performs comparably to state-of-the-art attitude estimation algorithms, such as the complementary filter or regular RNNs. By analyzing the network's response to manipulated sensor data, we found evidence that it effectively internalizes the underlying physical dynamics. For instance, when we replaced the accelerometer readings with a unit vector in the z-direction – simulating a scenario where only gravitational acceleration is measured – the

network exhibited long-term biases in the attitude estimate, as expected. Similarly, setting the gyroscope values to zero led to significantly slower responses during high-speed rotations, underscoring the importance of dynamic information in these measurements. Additionally, leveraging network sparsity significantly reduced the number of neuronal connections while retaining accuracy on the test data. Chapter 3 on the other hand, takes event-based vision as a source of information for estimating the attitude of the drone. De Croon *et al.* [46] showed that the combination of optical flow and a motion model allows for the estimation of attitude. In our study, we trained a recurrent neural network with camera events as inputs. Experiments with sensory data manipulations show that the network generalizes better when training on a cropped part of the view. Now, horizon-like appearance features along the periphery can no longer be exploited, forcing the network to model motion.

With the second research question, we wanted to delve deeper into the challenges that are introduced by including an SNN in the lower-level *control* loop of a drone:

> **Research Question 2**
>
> How can we design a network of spiking neurons to perform attitude control of a flying drone?

The term *flying* is important, since embedding neural controllers on a real platform leads to numerous challenges such as the reality gap and delays from hardware limitations. We demonstrated that networks of spiking neurons can be designed in various ways for attitude control. In Chapter 4, we show how neuron layers can be shaped for PID control using *position*-coding. This network was deployed on the Loihi neuromorphic processor to perform altitude control. Resolution remained an important challenge with this approach. Next, we trained groups of *rate*-coded neurons to perform PID control in Chapter 5. Additionally, we introduced a novel method for updating the threshold of neurons based on incoming spikes. This mechanism performs integration, rejecting constant disturbances and sensor bias. Learning the parameters that govern the adaptation proved difficult however, and perfect integration still required a tight connection between input encoding, network structure and neuron parameters.

We brought all of this together in Chapter 6 by answering Research Question 3:

> **Research Question 3**
>
> How can we create an SNN to perform full attitude estimation and control onboard?

Supervised learning of end-to-end control is extremely challenging. Neural networks often exploit unintended shortcuts in training data (e.g., spurious correlations) rather than learning the intended task structure (Geirhos *et al.* [130]). To mitigate this, we split the training into modular networks that can be efficiently connected through linear collapsing of their output and input weights. This was first introduced by Paredes-Vallés *et al.* [59] for connecting the outputs of a spiking optical flow estimator to a linear control layer, but also applies to connecting multiple spiking networks. To address integration with spiking neurons, in this chapter we proposed another method of integration: By fixing the leak parameters of a subgroup of neurons to 1, integration is achieved in the current that drives the membrane potential. Low-pass filtering the output spikes of these neurons then represents the integral. Apart from the fixed parameters, all the connections and neuron parameters could be learned via supervised learning.

Given these findings, we can confidently state that our research goal has been achieved:

> **Research Goal**
>
> Design and train neuromorphic algorithms that perform low-level flight control

Our research demonstrates that SNNs can function effectively as both estimators and controllers in real-world experiments, including successfully flying the Crazyflie drone from IMU input to motor command. Most importantly, in our efforts to achieve the research goal, we obtanied several insights into how SNNs should be set up and trained for control tasks. We also identified main challenges, which will be further discussed in Section 7.2.

7.2. Discussion

In this section, we will discuss some of the main challenges encountered and lessons learned. We will also project our conclusions on the future of the research field.

7.2.1. Fix parameters to fix gradient descent

One of the key challenges encountered during this work was the difficulty that supervised learning using gradient descent faces when approximating fundamental mathematical operations – such as integration and differentiation – that are critical for implementing a classical PID controller. Although both artificial neural networks (ANNs) and spiking neural networks (SNNs) are universal function approximators (e.g., as demonstrated by Hornik *et al.* [131] for ANNs and by Maass [132] and Zhang *et al.* [133] for SNNs), training them to perform these elementary yet precise operations remains non-trivial.

A classical PID regulator requires a single-step memory operation, where integration accumulates past error and differentiation calculates the rate of change of error. In their naive implementations, both operations demand a *perfect* retention of the previous time step's data: the integrator must accurately recall the accumulated error, while the differentiator must use the exact prior error value. Such precision is critical – any deviation in the integrator may fail to counteract constant disturbances or sensor biases, and errors in the differentiator can lead to an amplification of noise.

Our investigations in Chapters 5 and 6 revealed that relying entirely on standard architectures combined with gradient descent is insufficient. When training the network as a black box, the gradient-descent algorithm struggled to converge on the exact recurrent connections required to replicate the previous state perfectly. Overestimation of these connections resulted in an uncontrolled amplification of the hidden state, essentially pushing the recurrent gain above one and leading to network instability. Conversely, underestimation caused the system to behave merely as a low-pass filter, failing to capture the dynamics needed for proper control. To mitigate these issues, we adopted a strategy of fixing the parameters for specific subgroups of neurons. By constraining these parameters, we effectively ensured that the network initialized with regions in the search space that are critical for the precise replication of the previous time step—regions that gradient descent alone is unlikely to explore. This approach allowed the remainder of the network's parameters to adjust and optimize other aspects of the task, ultimately yielding improved performance in tasks such as PID control. This strategy not only enhanced the performance of our SNN implementations but also underscores a broader principle in deep learning: incorporating structured priors or fixed parameters can be instrumental when learning tasks that require precise mathematical operations. Physics-Informed

Machine Learning or Physics-Informed Neural Networks (PINNs) have emerged as a promising avenue, where embedding physical laws directly into the learning process serves as a natural regularizer by effectively "fixing" network structure to adhere to known dynamics (see Raissi *et al.* [134] and Karniadakis *et al.* [135]). Recent work on Structured State-Space Models (S4s) by Gu *et al.* [136] reflects a similar philosophy by demonstrating that treating a neural network as a dynamical system – using the fundamental state space model (SSM) – enables the accurate capture of long-range dependencies. Such an approach may be vital for enabling online learning and real-time robotic control in future systems.

In future work on this topic, it might also be interesting to look at circuitry found in nature to distill meaningful structure. Two examples of interesting neural circuitry are 1. the ring attractor dynamics in the *Drosophila* brain that seem to maintain a persistent and unique representation of its heading using local excitation and global inhibition (Kim *et al.* [137]) or 2. disinhibition (Letzkus *et al.* [138]), where transient reductions of inhibition might lead to long-lasting synaptic plasticity and thus allow memory to be formed.

7.2.2. On training spiking neural networks: beyond supervised learning

Although this thesis has primarily relied on supervised, surrogate gradient descent for parameter updates during training, several alternative approaches exist that may become highly relevant for future robotics applications. In fact, during this work we also explored multiple additional training methods in collaboration with fellow colleagues and Master's students.

In our first alternative approach, together with Tim Burgers, we investigated the use of evolutionary algorithms (EAs) to train spiking neural networks (SNNs) for PID control, as presented at IMAV 2023 [139]. This work showed that the parameters of IWTA and recurrent connections can be evolved from scratch to reduce steady-state errors, which was difficult to achieve with gradient descent – showcasing the potential of EAs to explore the solution space more effectively. However, the EAs shared difficulties with gradient descent in minimizing errors across multiple timescales simultaneously which was essential to enable fast derivative responses while also integrating errors over time, and perfect integration remained a challenge.

In the second approach, together with Korneel van den Berghe, we explored reinforcement learning (RL) as a method for end-to-end training of SNNs [140]. RL offers the benefit of eliminating the need for real-world data collection by using a

dynamic model of the robot during training. However, training recurrent networks (such as SNNs) with RL is inherently more challenging than training feedforward models due to temporal dependencies [141]. In SNNs, these dependencies arise from the propagation of current and voltage over time – a feature that endows them with temporal expressivity but also complicates the learning process. To address these challenges, we modified standard state-of-the-art RL methods, specifically the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm as outlined by Fujimoto *et al.* [142] and looked at scheduling of the surrogate-gradient slope. The modified RL was able to successfully learn how to map IMU sensor measurements to motor controls in simulations, and even allowed controlled flight of a real Crazyflie drone. However, the reality gap remains a significant barrier to real-world deployment. For instance, although the Crazyflie managed to fly with the RL-trained SNN, it still exhibited substantial oscillations around the setpoints. Ultimately, we expect that RL may allow for SNN-based control that outperforms classical control methods. In Song *et al.* [105], RL was shown to outperform state-of-the-art optimal control methods, thereby exposing some limitations of classical control approaches. Moreover, RL may offer pathways to achieve performance levels comparable to state-of-the-art controllers while requiring significantly less computational effort (Ferede *et al.* [118]).

Last, but certainly not least, we have looked at self-supervised learning methods. In two studies that I contributed to ([59] & [143]), we researched using contrast-maximization to determine optical flow and depth from an event-based camera respectively. The benefits of self-supervised learning are evident: No time- and money-consuming labeling of datasets is necessary and learning might be carried out while operating. In Paredes-Vallés *et al.* [59], an SNN is trained that estimates optical flow from an event-based camera. This SNN is then deployed on Intel's Loihi [19], producing estimates of the (scaled) velocity of the drone at a rate of 200Hz – extremely high compared to state-of-the-art vision pipelines. This was taken even further in Hagenaars *et al.* [143], where a depth image was learned from a forward-facing monocular event camera onboard at about 30Hz *while flying*. In the future, it would be valuable to explore the possibility for onboard, online self-supervised learning of attitude estimation and control as well.

7.2.3. The future of neuromorphic robotics: the chicken and the egg

While our work has demonstrated that SNNs can effectively serve in the lower-level control loop of a flying robot, advancements are still necessary before these systems can be deployed in fully task-ready drones. Looking forward, two intertwined fronts must be addressed: hardware development and algorithmic innovation. Although neuromorphic hardware holds great promise for enabling fully autonomous flight control, current systems face several significant challenges. First, the available neuromorphic processors suffer from communication bottlenecks: the limited number of input/output connections restricts the precision and scalability of the implemented SNNs, thereby hindering real-time performance. Moreover, most of these processors are still relatively large and must be interfaced with companion computers in order to connect them with sensors – which are also almost exclusively digital and synchronous. For a truly integrated neuromorphic flight controller, it is imperative to miniaturize these systems and eliminate external dependencies through better component integration. Also, research should be put toward replacing common flight sensors (such as the IMU) with neuromorphic alternatives. One example of processors that might be especially suited for SNNs are analog processors such as memristor crossbars (Li *et al.* [144]). Robotic innovation does not benefit from the large and powerful GPUs that can be found in datacenters and are currently the focus of massive tech-investments. Indeed, robotics will benefit from smaller, lightweight processors and sensors that can be easily integrated in a robotic system and satisfy very basic requirements. For example, in many navigational applications, there is little need for high-resolution cameras. Although such cameras can capture more detailed images, they typically generate significantly more data and demand greater computational power – factors that may not translate into tangible benefits for navigation tasks. In these scenarios, attributes such as a wide field-of-view are far more critical (van Dijk *et al.* [145]). However, developing specialized chips and sensors tailored for these applications is extremely expensive, and the current lack of clearly demonstrated, cost-effective applications limits industry incentives to invest in this area.

On the algorithmic front, we have already established that training SNNs remains a considerable challenge. The inherent difficulty of training these networks calls for new methods or improvements to existing ones. However, due to a lack of suitable hardware it is often difficult to accurately assess the benefits of the proposed methods. More often than not, a comparison is still made while

running on traditional synchronous hardware, which is always in favor of those algorithms that are designed for it. In summary, the path toward fully neuromorphic flight controllers hinges on simultaneous progress in both hardware and algorithms. While hardware advancements should provide the necessary platforms for efficient and compact neuromorphic systems, corresponding algorithmic breakthroughs are essential to unlock their full potential and demonstrate their transformative impact to industry. This mutual dependency creates a classic chicken-and-egg dilemma – each component is indispensable, yet progress in one area would propel advances in the other. Nonetheless, there is ample reason for optimism, as there is an abundance of both chickens and eggs all around us.

7.2.4. Concluding remarks

Our findings offer insights into integrating SNNs within broader hierarchical navigation and control frameworks. This paves the way for fully neuromorphic flight controllers operating on a single chip. In the near future, this could enable smaller, more efficient autonomous robots.

References

[1] S. Stroobants, J. Dupeyroux and G. C. de Croon. 'Neuromorphic computing for attitude estimation onboard quadrotors'. In: *Neuromorphic Computing and Engineering* 2.3 (2022), p. 034005.

[2] K. Rajan and A. Saffiotti. 'Towards a science of integrated AI and Robotics'. In: *Artificial Intelligence* 247 (June 2017), pp. 1–9. ISSN: 00043702. DOI: 10.1016/j.artint.2017.03.003.

[3] D. Ha and J. Schmidhuber. 'Recurrent world models facilitate policy evolution'. In: *Advances in neural information processing systems* 31 (2018).

[4] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik and S. Levine. 'Learning to poke by poking: Experiential learning of intuitive physics'. In: *Advances in neural information processing systems* 29 (2016).

[5] A. Giusti, J. Guzzi, D. C. Ciresan, F. L. He, J. P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza and L. M. Gambardella. 'A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots'. In: *IEEE Robotics and Automation Letters* 1 (2 July 2016), pp. 661–667. ISSN: 23773766. DOI: 10.1109/LRA.2015.2509024.

[6] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun and D. Scaramuzza. 'Learning high-speed flight in the wild'. In: *Science Robotics* 6.59 (2021), eabg5810.

[7] A. S. Poznyak, E. N. Sanchez and W. Yu. *Differential neural networks for robust nonlinear control: identification, state estimation and trajectory tracking*. World Scientific, 2001.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra. 'Continuous control with deep reinforcement learning'. In: *CoRR* abs/1509.02971 (2016).

[9] J. Hwangbo, I. Sa, R. Siegwart and M. Hutter. 'Control of a quadrotor with reinforcement learning'. In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 2096–2103.

[10] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy and T. Brox. 'Flownet 2.0: Evolution of optical flow estimation with deep networks'. In: *Proceedings*

of the IEEE conference on computer vision and pattern recognition. 2017, pp. 2462–2470.

[11] J. Redmon, S. Divvala, R. Girshick and A. Farhadi. 'You only look once: Unified, real-time object detection'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[12] N. K. Kasabov. *Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence*. 2019. URL: http://www.springer.com/series/15821.

[13] Y. Cao, Y. Chen and D. Khosla. 'Spiking deep convolutional neural networks for energy-efficient object recognition'. In: *International Journal of Computer Vision* 113.1 (2015), pp. 54–66.

[14] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier and S. Millner. 'A wafer-scale neuromorphic hardware system for large-scale neural modeling'. In: *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2010, pp. 1947–1950.

[15] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla and K. Boahen. 'Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations'. In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716.

[16] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.* 'A million spiking-neuron integrated circuit with a scalable communication network and interface'. In: *Science* 345.6197 (2014), pp. 668–673.

[17] S. B. Furber, F. Galluppi, S. Temple and L. A. Plana. 'The spinnaker project'. In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665.

[18] A. Calimera, E. Macii and M. Poncino. 'The human brain project and neuromorphic computing'. In: *Functional neurology* 28.3 (2013), p. 191.

[19] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.* 'Loihi: A neuromorphic manycore processor with on-chip learning'. In: *Ieee Micro* 38.1 (2018), pp. 82–99.

[20] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis *et al.* 'Event-based vision: A survey'. In: *IEEE transactions on pattern analysis and machine intelligence* 44.1 (2020), pp. 154–180.

[21] S. Caviglia, L. Pinna, M. Valle and C. Bartolozzi. 'An event-driven POSFET taxel for sustained and transient sensing'. In: *2016 IEEE International*

Symposium on Circuits and Systems (ISCAS). 2016, pp. 349–352. DOI: 10.1109/ISCAS.2016.7527242.

[22] A. Vitale, A. Renner, C. Nauer, D. Scaramuzza and Y. Sandamirskaya. 'Event-driven vision and control for UAVs on a neuromorphic chip'. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. Xi'an, China: IEEE, 2021, pp. 103–109. DOI: 10.1109/ICRA48506.2021.9560881.

[23] J. Dupeyroux, J. J. Hagenaars, F. Paredes-Vallés and G. C. de Croon. 'Neuromorphic control for optic-flow-based landing of MAVs using the Loihi processor'. In: *ICRA 2021: IEEE International Conference on Robotics and Automation*. IEEE. 2021, pp. 96–102.

[24] E. Rueckert, D. Kappel, D. Tanneberg, D. Pecevski and J. Peters. 'Recurrent Spiking Networks Solve Planning Tasks'. In: *Scientific Reports* 6 (Feb. 2016), p. 21142. DOI: 10.1038/srep21142.

[25] D. Weber, C. Gühmann and T. Seel. 'Neural networks versus conventional filters for inertial-sensor-based attitude estimation'. In: *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*. IEEE. 2020, pp. 1–8.

[26] R. Stagsted, A. Vitale, J. Binz, A. Renner, L. B. Larsen and Y. Sandamirskaya. 'Towards neuromorphic control: A spiking neural network based PID controller for UAV'. In: *Robotics: Science and Systems 2020, Virtual Conference*. RSS. Robotics: Science and Systems, June 2020. DOI: 10.15607/rss.2020.xvi.074.

[27] F. Furrer, M. Burri, M. Achtelik and R. Siegwart. 'RotorS—A modular gazebo MAV simulator framework'. In: *Studies in Computational Intelligence* 625 (Feb. 2016). Ed. by A. Koubaa, pp. 595–625. ISSN: 1860949X. DOI: 10.1007/978-3-319-26054-9_23.

[28] C. Pehle and J. E. Pedersen. *Norse - A deep learning library for spiking neural networks*. Version 0.0.6. Documentation: https://norse.ai/docs/. Jan. 2021. DOI: 10.5281/zenodo.4422025. URL: https://doi.org/10.5281/zenodo.4422025.

[29] D. Kingma and J. Ba. 'Adam: A Method for Stochastic Optimization'. In: *International Conference on Learning Representations* (Dec. 2014).

[30] M. Zhang, J. Lucas, J. Ba and G. E. Hinton. 'Lookahead optimizer: k steps forward, 1 step back'. In: *Advances in Neural Information Processing Systems* 32 (2019).

[31] E. Neftci, H. Mostafa and F. Zenke. 'Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to

Spiking Neural Networks'. In: *IEEE Signal Processing Magazine* 36 (Nov. 2019), pp. 51–63. DOI: 10.1109/MSP.2019.2931595.

[32] F. Zenke and S. Ganguli. 'SuperSpike: Supervised learning in multilayer spiking neural networks'. In: *Neural Computation* 30 (6 June 2018), pp. 1514–1541. ISSN: 1530888X. DOI: 10.1162/neco_a_01086.

[33] F. Zenke and T. P. Vogels. 'The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks'. In: *Neural Computation* 33.4 (2021), pp. 899–925.

[34] S. O. Madgwick, A. J. Harrison and R. Vaidyanathan. 'Estimation of IMU and MARG orientation using a gradient descent algorithm'. In: *IEEE International Conference on Rehabilitation Robotics* (2011), pp. 179–185.

[35] R. Mahony, T. Hamel and J.-M. Pflimlin. 'Nonlinear complementary filters on the special orthogonal group'. In: *IEEE Transactions on automatic control* 53.5 (2008), pp. 1203–1218.

[36] R. E. Kalman. 'A new approach to linear filtering and prediction problems'. In: (1960).

[37] M. I. Ribeiro. 'Kalman and extended kalman filters: Concept, derivation and properties'. In: *Institute for Systems and Robotics* 43 (2004), p. 46.

[38] P. Gui, L. Tang and S. Mukhopadhyay. 'MEMS based IMU for tilting measurement: Comparison of complementary and kalman filter based data fusion'. In: Institute of Electrical and Electronics Engineers Inc., Nov. 2015, pp. 2004–2009. ISBN: 9781467373173. DOI: 10.1109/ICIEA.2015.7334442.

[39] Y. Shi and R. Eberhart. 'A modified particle swarm optimizer'. In: *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*. IEEE. 1998, pp. 69–73.

[40] A. M. Sabatini. 'Kalman-filter-based orientation determination using inertial/magnetic sensors: Observability analysis and performance evaluation'. In: *Sensors* 11 (10 Oct. 2011), pp. 9182–9206. ISSN: 14248220. DOI: 10.3390/s111009182.

[41] T.-A. Johansen and R. Kristiansen. 'Quadrotor attitude estimation using adaptive fading multiplicative EKF'. In: *2017 American Control Conference (ACC)*. IEEE. 2017, pp. 1227–1232.

[42] J. J. Hagenaars, S. Stroobants, S. M. Bohte and G. C. De Croon. 'All Eyes, no IMU: Learning Flight Attitude from Vision Alone'. In: *arXiv preprint arXiv:2507.11302* (2025).

[43] R. Mahony, T. Hamel and J.-M. Pflimlin. 'Nonlinear Complementary Filters on the Special Orthogonal Group'. In: *IEEE Transactions on Automatic Control* 53 (2008), pp. 1203–1218. DOI: 10.1109/TAC.2008.923738.

[44] G. K. Taylor and H. G. Krapp. 'Sensory Systems and Flight Stability: What Do Insects Measure and Why?' In: *Advances in Insect Physiology*. Vol. 34. Insect Mechanics and Control. London: Academic Press, 2007, pp. 231–316. DOI: 10.1016/S0065-2806(07)34005-8.

[45] M. V. Srinivasan, R. J. D. Moore, S. Thurrowgood, D. Soccol and D. Bland. 'From Biology to Engineering: Insect Vision and Applications to Robotics'. In: *Frontiers in Sensing*. Vienna: Springer, 2012, pp. 19–39. DOI: 10.1007/978-3-211-99749-9_2.

[46] G. C. De Croon, J. J. Dupeyroux, C. De Wagter, A. Chatterjee, D. A. Olejnik and F. Ruffier. 'Accommodating unobservability to control flight attitude with optic flow'. In: *Nature* 610.7932 (2022), pp. 485–490.

[47] Z. Yu, J. Tran, C. Li, A. Weber, Y. P. Talwekar and S. Fuller. *TinySense: A Lighter Weight and More Power-efficient Avionics System for Flying Insect-scale Robots*. 2025. DOI: 10.48550/arXiv.2501.03416. arXiv: 2501.03416 [cs].

[48] S. M. Ettinger, M. C. Nechyba, P. G. Ifju and M. Waszak. 'Vision-guided flight stability and control for micro air vehicles'. In: *Advanced Robotics* 17.7 (2003), pp. 617–640.

[49] I. F. Mondragón, M. A. Olivares-Méndez, P. Campoy, C. Martínez and L. Mejias. 'Unmanned aerial vehicles UAVs attitude, height, motion estimation and control using visual systems'. In: *Autonomous Robots* 29 (2010), pp. 17–34.

[50] J.-C. Bazin, I. Kweon, C. Demonceaux and P. Vasseur. 'UAV attitude estimation by vanishing points in catadioptric images'. In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 2743–2749.

[51] A. E. R. Shabayek, C. Demonceaux, O. Morel and D. Fofi. 'Vision based uav attitude estimation: Progress and insights'. In: *Journal of Intelligent & Robotic Systems* 65 (2012), pp. 295–308.

[52] G. Gallego and D. Scaramuzza. 'Accurate angular velocity estimation with an event camera'. In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 632–639.

[53] M. Gehrig, S. B. Shrestha, D. Mouritzen and D. Scaramuzza. 'Event-based angular velocity regression with spiking networks'. In: *2020 IEEE*

International Conference on Robotics and Automation (ICRA). IEEE. 2020, pp. 4195–4202.

[54] H. Greatorex, M. Mastella, M. Cotteret, O. Richter and E. Chicca. 'Event-based vision for egomotion estimation using precise event timing'. In: *arXiv preprint arXiv:2501.11554* (2025).

[55] D. R. Da Costa, P. Vasseur and F. Morbidi. 'Gyrevento: Event-based Omnidirectional Visual Gyroscope in a Manhattan World'. In: *IEEE Robotics and Automation Letters* (2025), pp. 1–8. DOI: 10.1109/LRA.2025.3527311.

[56] I. Geles, L. Bauersfeld, A. Romero, J. Xing and D. Scaramuzza. 'Demonstrating Agile Flight from Pixels without State Estimation'. In: *Robotics: Science and Systems XX*. Vol. 20. 2024.

[57] J. Xing, A. Romero, L. Bauersfeld and D. Scaramuzza. 'Bootstrapping reinforcement learning with imitation for vision-based agile flight'. In: *arXiv preprint arXiv:2403.12203* (2024).

[58] A. Romero, A. Shenai, I. Geles, E. Aljalbout and D. Scaramuzza. *Dream to Fly: Model-Based Reinforcement Learning for Vision-Based Drone Flight*. 2025. DOI: 10.48550/arXiv.2501.14377. arXiv: 2501.14377 [cs].

[59] F. Paredes-Vallés, J. Hagenaars, J. Dupeyroux, S. Stroobants, Y. Xu and G. de Croon. 'Fully neuromorphic vision and control for autonomous drone flight'. In: *arXiv preprint arXiv:2303.08778* 9.90 (2023), eadi0591. DOI: 10.1126/scirobotics.adi0591. URL: https://www.science.org/doi/abs/10.1126/scirobotics.adi0591.

[60] S. Stroobants, C. De Wagter and G. C. H. E. de Croon. 'Neuromorphic Attitude Estimation and Control'. In: *IEEE Robotics and Automation Letters* 10 (2025), pp. 4858–4865. DOI: 10.1109/LRA.2025.3553418.

[61] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck and D. Scaramuzza. 'The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM'. In: *The International journal of robotics research* 36.2 (2017), pp. 142–149.

[62] G. Gallego, H. Rebecq and D. Scaramuzza. 'A Unifying Contrast Maximization Framework for Event Cameras, With Applications to Motion, Depth, and Optical Flow Estimation'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3867–3876.

[63] O. Richter, Y. Xing, M. De Marchi, C. Nielsen, M. Katsimpris, R. Cattaneo, Y. Ren, Y. Hu, Q. Liu, S. Sheik *et al.* 'Speck: A smart event-based vision sensor

with a low latency 327k neuron convolutional neuronal network processing pipeline'. In: *arXiv preprint arXiv:2304.06793* (2023).

[64] D.-A. Clevert, T. Unterthiner and S. Hochreiter. 'Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)'. In: *International Conference on Learning Representations*. 2016. DOI: 10.48550/arXiv.1511.07289. eprint: 1511.07289 (cs).

[65] S. Macenski, T. Foote, B. Gerkey, C. Lalancette and W. Woodall. 'Robot operating system 2: Design, architecture, and uses in the wild'. In: *Science robotics* 7.66 (2022), eabm6074.

[66] S. Stroobants, J. Dupeyroux and G. De Croon. 'Design and implementation of a parsimonious neuromorphic PID for onboard altitude control for MAVs using neuromorphic processors'. In: *Proceedings of the International Conference on Neuromorphic Systems 2022*. Knoxville, TN, USA: Association for Computing Machinery, 2022, pp. 1–7. DOI: 10.1145/3546790.3546799.

[67] D. Floreano and R. J. Wood. 'Science, technology and the future of small autonomous drones'. In: *Nature* 521.7553 (2015), pp. 460–466.

[68] Z. Zheng, J. S. Lauritzen, E. Perlman, C. G. Robinson, M. Nichols, D. Milkie, O. Torrens, J. Price, C. B. Fisher, N. Sharifi *et al.* 'A complete electron microscopy volume of the brain of adult Drosophila melanogaster'. In: *Cell* 174.3 (2018), pp. 730–743.

[69] M. Müller and R. Wehner. 'Path integration in desert ants, Cataglyphis fortis'. In: *Proceedings of the National Academy of Sciences* 85.14 (1988), pp. 5287–5290.

[70] S. Jung, S. Hwang, H. Shin and D. H. Shim. 'Perception, guidance, and navigation for indoor autonomous drone racing using deep learning'. In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2539–2544.

[71] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun and D. Scaramuzza. 'Beauty and the beast: Optimal methods meet learning for drone racing'. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 690–696.

[72] M. Lundstrom. 'Moore's law forever?' In: *Science* 299.5604 (2003), pp. 210–211.

[73] T. N. Theis and H.-S. P. Wong. 'The end of moore's law: A new beginning for information technology'. In: *Computing in Science & Engineering* 19.2 (2017), pp. 41–50.

[74] M. Mahowald and R. Douglas. 'A silicon neuron'. In: *Nature* 354.6354 (1991), pp. 515–518.

[75] S. Ghosh-Dastidar and H. Adeli. 'Spiking neural networks'. In: *International journal of neural systems* 19.04 (2009), pp. 295–308.

[76] A. Jimenez-Fernandez, G. Jimenez-Moreno, A. Linares-Barranco, M. J. Dominguez-Morales, R. Paz-Vicente and A. Civit-Balcells. 'A Neuro-Inspired Spike-Based PID Motor Controller for Multi-Motor Robots with Low Cost FPGAs'. In: *Sensors* 12.4 (2012), pp. 3831–3856. ISSN: 1424-8220. DOI: 10.3390/s120403831. URL: https://www.mdpi.com/1424-8220/12/4/3831.

[77] F. Perez-Peña, A. Morgado-Estevez, A. Linares-Barranco, A. Jimenez-Fernandez, F. Gomez-Rodriguez, G. Jimenez-Moreno and J. Lopez-Coronado. 'Neuro-Inspired Spike-Based Motion: From Dynamic Vision Sensor to Robot Motor Open-Loop Control through Spike-VITE'. In: *Sensors* 13.11 (2013), pp. 15805–15832. ISSN: 1424-8220. DOI: 10.3390/s131115805. URL: https://www.mdpi.com/1424-8220/13/11/15805.

[78] T. Dewolf, T. Stewart, J.-J. Slotine and C. Eliasmith. 'A spiking neural model of adaptive arm control'. In: *Proceedings of the Royal Society of London B: Biological Sciences* 283 (Nov. 2016). DOI: 10.1098/rspb.2016.2134.

[79] Y. Zaidel, A. Shalumov, A. Volinski, L. Supic and E. E. Tsur. 'Neuromorphic NEF-Based Inverse Kinematics and PID Control'. In: *Frontiers in Neurorobotics* 15.631159 (Feb. 2021), p. 2. ISSN: 1662-5218. DOI: 10.3389/fnbot.2021.631159.

[80] K. J. Åström and B. Wittenmark. *Computer-controlled systems: theory and design*. Courier Corporation, 2013.

[81] S. Lynen, M. Achtelik, S. Weiss, M. Chli and R. Siegwart. 'A Robust and Modular Multi-Sensor Fusion Approach Applied to MAV Navigation'. In: *Proc. of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 2013.

[82] C.-K. Lin, A. Wild, G. N. Chinya, Y. Cao, M. Davies, D. M. Lavery and H. Wang. 'Programming spiking neural networks on Intel's Loihi'. In: *Computer* 51.3 (2018), pp. 52–61.

[83] S. Stroobants, C. De Wagter and G. De Croon. 'Neuromorphic Control using Input-Weighted Threshold Adaptation'. In: *Proceedings of the 2023 International Conference on Neuromorphic Systems*. 2023, pp. 1–8.

[84] P. R. Wurman, R. D'Andrea and M. Mountz. 'Coordinating hundreds of cooperative, autonomous vehicles in warehouses'. In: *AI magazine* 29.1 (2008), pp. 9–9.

[85] S. Sekander, H. Tabassum and E. Hossain. 'Multi-tier drone architecture for 5G/B5G cellular networks: Challenges, trends, and prospects'. In: *IEEE Communications Magazine* 56.3 (2018), pp. 96–103.

[86] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose and J. S. Plank. 'A survey of neuromorphic computing and neural networks in hardware'. In: *arXiv preprint arXiv:1705.06963* (2017).

[87] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro *et al.* 'A machine learning approach to visual perception of forest trails for mobile robots'. In: *IEEE Robotics and Automation Letters* 1.2 (2015), pp. 661–667.

[88] I. Abadía, F. Naveros, E. Ros, R. R. Carrillo and N. R. Luque. 'A cerebellar-based solution to the nondeterministic time delay problem in robotic control'. In: *Science Robotics* 6.58 (2021).

[89] M. H. Dickinson. 'Haltere-mediated equilibrium reflexes of the fruit fly, Drosophila melanogaster'. In: *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences* 354.1385 (1999), pp. 903–916.

[90] T. S. Clawson, S. Ferrari, S. B. Fuller and R. J. Wood. 'Spiking neural network (SNN) control of a flapping insect-scale robot'. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE. Las Vegas, NV, USA: IEEE, 2016, pp. 3381–3388. DOI: 10.1109/CDC.2016.7798778.

[91] H. Qiu, M. Garratt, D. Howard and S. Anavatti. 'Evolving spiking neurocontrollers for UAVs'. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. Canberra, ACT, Australia: IEEE, 2020, pp. 1928–1935. DOI: 10.1109/SSCI47803.2020.9308275.

[92] M. Levakova, L. Kostal, C. Monsempès, P. Lucas and R. Kobayashi. 'Adaptive integrate-and-fire model reproduces the dynamics of olfactory receptor neuron responses in a moth'. In: *Journal of The Royal Society Interface* 16.157 (2019), p. 20190246. DOI: 10.1098/rsif.2019.0246. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rsif.2019.0246.

[93] R. Brette and W. Gerstner. 'Adaptive exponential integrate-and-fire model as an effective description of neuronal activity'. In: *Journal of neurophysiology* 94.5 (2005), pp. 3637–3642.

[94] F. Paredes-Vallés, K. Y. Scheper and G. C. De Croon. 'Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: From events to global motion perception'. In: *IEEE transactions on pattern analysis and machine intelligence* 42.8 (2019), pp. 2051–2064.

[95] K. J. Astrom and L. Rundqwist. 'Integrator windup and how to avoid it'. In: *1989 American Control Conference*. IEEE. Pittsburgh, PA, USA: IEEE, 1989, pp. 1693–1698. DOI: 10.23919/ACC.1989.4790464.

[96] G. Di Chiara, M. Morelli and S. Consolo. 'Modulatory functions of neurotransmitters in the striatum: ACh/dopamine/NMDA interactions'. In: *Trends in neurosciences* 17.6 (1994), pp. 228–233.

[97] E. O. Neftci, H. Mostafa and F. Zenke. 'Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks'. In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 51–63.

[98] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang and Y. Tian. 'Incorporating learnable membrane time constant to enhance learning of spiking neural networks'. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. Virutal: CVPR, 2021, pp. 2661–2671.

[99] I. Cohen, Y. Huang, J. Chen, J. Benesty, J. Benesty, J. Chen, Y. Huang and I. Cohen. 'Pearson Correlation Coefficient'. In: *Noise Reduction in Speech Processing*. Berlin, Heidelberg: Springer, 2009, pp. 1–4. ISBN: 978-3-642-00296-0. DOI: 10.1007/978-3-642-00296-0_5.

[100] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński and P. Kozierski. 'Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering'. In: *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE. Miedzyzdroje, Poland: IEEE, 2017, pp. 37–42. DOI: 10.1109/MMAR.2017.8046794.

[101] B. Han, G. Srinivasan and K. Roy. 'Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network'. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. Virtual: CVPR, 2020, pp. 13558–13567.

[102] S. M. S. M. Daud, M. Y. P. M. Yusof, C. C. Heo, L. S. Khoo, M. K. C. Singh, M. S. Mahmood and H. Nawawi. 'Applications of drone in disaster management: A scoping review'. In: *Science & Justice* 62.1 (2022), pp. 30–42.

[103] L. Tang and G. Shao. 'Drone remote sensing for forestry research and practices'. In: *Journal of forestry research* 26 (2015), pp. 791–797.

[104] U. R. Mogili and B. Deepak. 'Review on application of drone systems in precision agriculture'. In: *Procedia computer science* 133 (2018), pp. 502–509.

[105] Y. Song, A. Romero, M. Müller, V. Koltun and D. Scaramuzza. 'Reaching the limit in autonomous racing: Optimal control versus reinforcement learning'. In: *Science Robotics* 8.82 (2023), eadg1462. DOI: 10.1126/scirobotics.adg1462. URL: https://www.science.org/doi/abs/10.1126/scirobotics.adg1462.

[106] A. T. Azar, A. Koubaa, N. Ali Mohamed, H. A. Ibrahim, Z. F. Ibrahim, M. Kazim, A. Ammar, B. Benjdira, A. M. Khamis, I. A. Hameed *et al.* 'Drone deep reinforcement learning: A review'. In: *Electronics* 10.9 (2021), p. 999.

[107] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford *et al.* 'The limits and potentials of deep learning for robotics'. In: *The International journal of robotics research* 37.4-5 (2018), pp. 405–420.

[108] Y. Sandamirskaya, M. Kaboli, J. Conradt and T. Celikel. 'Neuromorphic computing hardware and neural architectures for robotics'. In: *Science Robotics* 7.67 (2022), eabl8419. DOI: 10.1126/scirobotics.abl8419. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abl8419.

[109] P. Lichtsteiner, C. Posch and T. Delbruck. 'A 128×128 120 dB 15μs latency asynchronous temporal contrast vision sensor'. In: *IEEE Journal of Solid-State Circuits* 43.2 (2008), pp. 566–576.

[110] G. Indiveri and R. Douglas. 'Neuromorphic vision sensors'. In: *Science* 288.5469 (2000), pp. 1189–1190.

[111] W. Maass. 'Networks of spiking neurons: the third generation of neural network models'. In: *Neural Networks* 10.9 (1997), pp. 1659–1671.

[112] C. Bartolozzi, G. Indiveri and E. Donati. 'Embodied neuromorphic intelligence'. In: *Nature Communications* 13.1 (2022), p. 1024.

[113] R. Pellerito, M. Cannici, D. Gehrig, J. Belhadj, O. Dubois-Matra, M. Casasco and D. Scaramuzza. 'End-to-End Learned Event-and Image-based Visual Odometry'. In: *arXiv preprint arXiv:2309.09947* (2023).

[114] R. S. Dimitrova, M. Gehrig, D. Brescianini and D. Scaramuzza. 'Towards low-latency high-bandwidth control of quadrotors using event cameras'. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 4294–4300.

[115] D. Falanga, K. Kleber and D. Scaramuzza. 'Dynamic obstacle avoidance for quadrotors with event cameras'. In: *Science Robotics* 5.40 (2020), eaaz9712. DOI: 10.1126/scirobotics.aaz9712. URL: https://www.science.org/doi/abs/10.1126/scirobotics.aaz9712.

[116] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun and D. Scaramuzza. 'Champion-level drone racing using deep reinforcement learning'. In: *Nature* 620.7976 (2023), pp. 982–987.

[117] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun and D. Scaramuzza. 'Deep Drone Acrobatics'. In: *Proceedings of Robotics: Science and Systems*. Corvalis, Oregon, USA, July 2020. DOI: 10.15607/RSS.2020.XVI.040.

[118] R. Ferede, C. De Wagter, D. Izzo and G. C. De Croon. 'End-to-end reinforcement learning for time-optimal quadcopter flight'. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 6172–6177.

[119] T. Zhang, G. Kahn, S. Levine and P. Abbeel. 'Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search'. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 528–535.

[120] F. S. Slijkhuis, S. W. Keemink and P. Lanillos. 'Closed-form control with spike coding networks'. In: *IEEE Transactions on Cognitive and Developmental Systems* (2023).

[121] P. Martin and E. Salaün. 'The true role of accelerometer feedback in quadrotor control'. In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 1623–1629.

[122] S. Ross, G. Gordon and D. Bagnell. 'A reduction of imitation learning and structured prediction to no-regret online learning'. In: *Proceedings of the fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 627–635.

[123] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell and M. Hebert. 'Learning monocular reactive uav control in cluttered natural environments'. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 1765–1772.

[124] A. Orvieto, S. L. Smith, A. Gu, A. Fernando, C. Gulcehre, R. Pascanu and S. De. 'Resurrecting recurrent neural networks for long sequences'. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 26670–26698.

[125] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang and T. Januschowski. 'Deep state space models for time series forecasting'. In: *Advances in neural information processing systems* 31 (2018).

[126] H. Yu, G. C. E. de Croon and C. De Wagter. 'Avoidbench: A high-fidelity vision-based obstacle avoidance benchmarking suite for multi-rotors'. In: *2023*

IEEE International Conference on Robotics and Automation (ICRA). IEEE. 2023, pp. 9183–9189.

[127] G. Orchard, E. P. Frady, D. B. D. Rubin, S. Sanborn, S. B. Shrestha, F. T. Sommer and M. Davies. 'Efficient neuromorphic signal processing with loihi 2'. In: *2021 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE. 2021, pp. 254–259.

[128] H. Luo and W. Sun. 'Addition is All You Need for Energy-efficient Language Models'. In: *arXiv preprint arXiv:2410.00907* (2024).

[129] J.-F. Guerrero-Castellanos, J. J. Téllez-Guzmán, S. Durand, N. Marchand, J. U. Alvarez-Muñoz and V. R. Gonzalez-Diaz. 'Attitude stabilization of a quadrotor by means of event-triggered nonlinear control'. In: *Journal of Intelligent & Robotic Systems* 73 (2014), pp. 123–135.

[130] R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge and F. A. Wichmann. 'Shortcut learning in deep neural networks'. In: *Nature Machine Intelligence* 2.11 (2020), pp. 665–673.

[131] K. Hornik, M. Stinchcombe and H. White. 'Multilayer feedforward networks are universal approximators'. In: *Neural networks* 2.5 (1989), pp. 359–366.

[132] W. Maass. 'Lower bounds for the computational power of networks of spiking neurons'. In: *Neural computation* 8.1 (1996), pp. 1–40.

[133] S.-Q. Zhang and Z.-H. Zhou. 'Theoretically provable spiking neural networks'. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 19345–19356.

[134] M. Raissi, P. Perdikaris and G. E. Karniadakis. 'Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations'. In: *Journal of Computational physics* 378 (2019), pp. 686–707.

[135] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang and L. Yang. 'Physics-informed machine learning'. In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440.

[136] A. Gu, K. Goel and C. Ré. 'Efficiently modeling long sequences with structured state spaces'. In: *arXiv preprint arXiv:2111.00396* (2021).

[137] S. S. Kim, H. Rouault, S. Druckmann and V. Jayaraman. 'Ring attractor dynamics in the Drosophila central brain'. In: *Science* 356.6340 (2017), pp. 849–853.

[138] J. J. Letzkus, S. B. Wolff and A. Lüthi. 'Disinhibition, a circuit mechanism for associative learning and memory'. In: *Neuron* 88.2 (2015), pp. 264–276.

[139] T. Burgers, S. Stroobants and G. de Croon. 'Evolving Spiking Neural Networks to Mimic PID Control for Autonomous Blimps'. In: *arXiv preprint arXiv:2309.12937* (2023).

[140] K. Van den Berghe, S. Stroobants and G. De Croon. 'Adaptive Surrogate Gradients for Sequential Reinforcement Learning in Spiking Neural Networks'. In: *Advances in Neural Information Processing Systems* 37 (2025).

[141] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos and W. Dabney. 'Recurrent experience replay in distributed reinforcement learning'. In: *International conference on learning representations*. 2018.

[142] S. Fujimoto, H. Hoof and D. Meger. 'Addressing function approximation error in actor-critic methods'. In: *International conference on machine learning*. PMLR. 2018, pp. 1587–1596.

[143] J. J. Hagenaars, Y. Wu, F. Paredes-Vallés, S. Stroobants and G. C. de Croon. 'On-Device Self-Supervised Learning of Low-Latency Monocular Depth from Only Events'. In: *Proceedings of the Computer Vision and Pattern Recognition Conference*. 2025, pp. 17114–17123.

[144] C. Li, M. Hu, Y. Li, H. Jiang, N. Ge, E. Montgomery, J. Zhang, W. Song, N. Dávila, C. E. Graves *et al.* 'Analogue signal and image processing with large memristor crossbars'. In: *Nature electronics* 1.1 (2018), pp. 52–59.

[145] T. van Dijk, C. De Wagter and G. C. de Croon. 'Visual route following for tiny autonomous robots'. In: *Science Robotics* 9.92 (2024), eadk0310.

Acknowledgements

Life becomes bearable only when one has come to terms with who one is, both in one's own eyes and in the eyes of the world.

Sándor Márai, Embers

The search for some kind of identity—internal or external—often takes hold of me. I wonder who I am, who I want to be and who I should be, and often these three collide or I mistake one for another. Life is full of so many intriguing things and skills to dig into, but opening one door inevitably closes another. As those close to me likely know well, this often draws me into uncertainty and doubt. Amid all this, I am extremely lucky that I have you, Pauline, to support me. You show me the direction that lies somewhere under all the chaos in my head and I am glad that we can travel our path together.

I am thankful to Guido de Croon for pulling me back to university to further explore robotics and AI (and myself), after working in industry for a little over a year. I am impressed by how quickly you grasp the ideas and literature that I have pushed unto you over the last couple of years and I will always remember how much you care for the students under your supervision. Even though I often still find the Dutch-proverbial 'bears' on my road, you have helped me successfully identify and neutralize them. Whenever drones refused to behave, Christophe de Wagter's unparalleled insights in drone dynamics and electronics brought clarity, and his no-nonsense approach kept me grounded. For that, I am deeply grateful.

Beyond my supervisors, the MAVLab as a whole provided the environment that made this journey possible and I want to express how much I have loved working with you all. We put collaboration over internal competition and that is what leads to amazing results, in scientific achievements, international competitions and personal connections. Please never change. Most of all, my office-mates Jesse, Hang, and Yilun have shaped me and my scientific work. I have learned so much from you three and am grateful that we ended up in the same office (even when policies tried to separate us). It was humbling to share a room with people

that are as smart and ambitious as you guys, but that gave me the necessary motivation to keep going.

I am a firm believer of the significance of nurture over nature, and see the importance of sharing in the lives of those around me. You all have shaped me in many ways.

I am really fortunate with my dear friends and old roommates from OD119. Living with you all gave me great insights and I am glad I can still be a part of your lives today. The monthly "Spellentafel", TTRPG's, New Year's celebrations (LOTR-marathon awaits us, again), festivals or random meet-ups have really *"geschept"* a profound happiness in me. Also thank you, Paul$_a$, for your amazing help with the design of this thesis. Your eye for detail is indisputable and unmatched.

The people I met at Laga have also brightened up my life, especially those from the DSRVMG. I am glad that even after 2 "lustrums" we still manage to find time for weekends away or a ski-trip. *"Oh, ik ben rijker dan ik ooit heb durven dromen"*. And Joël; often melancholy seizes me when I pass "de Lelie" and I see other people enjoying their ice-cream along the canal.

Tony and Paul$_b$, thanks for listening to me rambling on about everything and nothing and for shouting incomprehensibly at me from time to time. I would not share my "diepvriespizza" with anyone else.

Kiefer, thank you for showing me what friendship means, for being so extremely generous to anyone, defining the meaning of comfort food and for sometimes pushing me over an edge when I need it. Along with Chun, Eric, Jaïr and everyone else that identifies as a shark neck, you have ensured a perfect start to almost every weekend over my PhD years by hosting Ristorante.

I want to thank my mother, Jacqueline, and father, Frank and my sister Bente. Jullie zijn er altijd wanneer nodig, ondersteunen me, maar nooit dwingend. De afgelopen jaren waren helaas ook erg zwaar en alle drie hebben jullie mij op eigen manier laten zien hoe we hier mee om kunnen gaan. Het leven is kort, soms heel pijnlijk, maar het zeker waard.

Frans, jij hebt me geleerd de oorsprong te vinden van mijn overtuigingen en perspectief. Dat is soms een lastige, confronterende zoektocht, maar niet één die men uit de weg moet gaan.

Now, reaching the end of this dissertation, it feels time to gently close this door—thankful that the band-aid comes off in a steady way called a "PostDoc"—and curious about the doors waiting to open after winter.

Curriculum Vitæ

Stein Stroobants

21-03-1995 Born in Amsterdam, The Netherlands.

Education

2007–2013	Secondary School Vossius Gymnasium, Amsterdam
2013–2016	BSc. Civil Engineering Delft University of Technology
2016–2017	Bridging program Mechanical Engineering Delft University of Technology
2017–2019	MSc. Control Theory Delft University of Technology
2021–2025	PhD. Aerospace Engineering Delft University of Technology *Thesis:* Neuromorphic Autopilot for Drone Flight *Promotor:* Prof. dr. G.C.H.E. De Croon

List of Publications

Journal Papers

4. **S. Stroobants**, J.J. Hagenaars, S. Bohté, G.C.H.E. de Croon (2025). All eyes, no IMU: Learning Flight Attitude from Events Alone. *Under Review*.

3. **S. Stroobants**, C. de Wagter, and De Croon, G. C. (2025). Neuromorphic Attitude Estimation and Control. Robotics and Automation Letters, vol. 10, no. 5, pp. 4858-4865

2. F. Paredes-Vallés, J.J. Hagenaars, J. Dupeyroux, **S. Stroobants**, Y. Xu and G.C.H.E. de Croon (2024). Fully neuromorphic vision and control for autonomous drone flight. *Science Robotics*, *9*(90), eadi0591.

1. **S. Stroobants**, J. Dupeyroux, and G.C.H.E. de Croon (2022). Neuromorphic computing for attitude estimation onboard quadrotors. *Neuromorphic Computing and Engineering*, 2(3), 034005.

Conference Papers

7. [1]K. Van den Berghe, **S. Stroobants**, V.J. Reddi, G.C.H.E. de Croon (2025). Adaptive Surrogate Gradients for Sequential Reinforcement Learning in Spiking Neural Networks. *Advances in Neural Information Processing Systems (NeurIPS)*.

6. [2]J.J. Hagenaars, Y. Wu, F. Paredes-Vallés, **S. Stroobants** and G.C.H.E. de Croon (2025). On-Device Self-Supervised Learning of Low-Latency Monocular Depth from Only Events. *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, pp. 17114-17123.

5. T. Burgers, **S. Stroobants** and G.C.H.E. de Croon (2024). Evolving Spiking Neural Networks to Mimic PID Control for Autonomous Blimps. *15th Annual*

[1]Selected for Oral Presentation at NeurIPS 2025
[2]Awarded with Best Paper Award at the NeuRobots 2025 IROS Workshop

International Micro Air Vehicle Conference and Competition (IMAV), p. 73–79.

4. **S. Stroobants**, C. De Wagter and G.C.H.E. De Croon (2023). Neuromorphic Control using Input-Weighted Threshold Adaptation. In *Proceedings of the 2023 International Conference on Neuromorphic Systems (ICONS)* (pp. 1-8).

3. J. Dupeyroux, **S. Stroobants** and G.C.H.E De Croon (2022). A toolbox for neuromorphic perception in robotics. In *2022 8th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)* (pp. 1-7). IEEE.

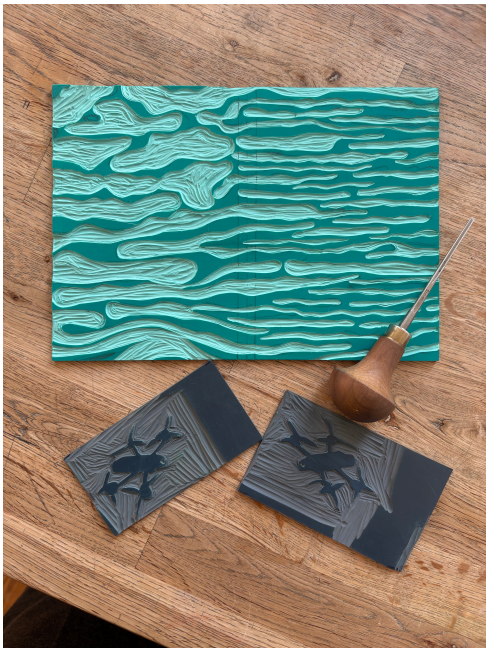
2. [3]**S. Stroobants**, J. Dupeyroux and G.C.H.E De Croon (2022). Design and implementation of a parsimonious neuromorphic PID for onboard altitude control for MAVs using neuromorphic processors. In *Proceedings of the 2022 International Conference on Neuromorphic Systems (ICONS)* (pp. 1-7).

1. D.A. Olejnik, S. Wang, J. Dupeyroux, **S. Stroobants**, M. Karasek, C. De Wagter and G.C.H.E de Croon (2022). An experimental study of wind resistance and power consumption in mavs with a low-speed multi-fan wind system. In *2022 International Conference on Robotics and Automation (ICRA)* (pp. 2989-2994). IEEE.

[3]Awarded with *Best Student Paper Award at ICONS 2025*

About the cover

My mother is an amazing artist, and I'm deeply honored that she designed the cover for this thesis. She showed incredible patience with my many requests. You can find her work at `www.jacquelino.nl` or on Instagram (@jacqsteinhoff). I especially recommend her still lifes — she captures the essence of a scene beautifully with just a few layers of color in her reduction prints.



One of my favorites on her website: